

به نام خدا



آزمایشگاه سخت افزار  
پروژه تحلیل ثابت افزار

دانشکده مهندسی کامپیوتر  
دانشگاه صنعتی شریف

---

نام، نام خانوادگی و شماره دانشجویی اعضای گروه:  
فرزام زهدی نسب - ۹۷۱۰۵۹۹۶  
دالیا داودی - ۹۷۱۱۰۳۹۳  
علی بالاپور - ۹۷۱۰۱۳۲۶

## فهرست مطالب

۳	۱	<b>مقدمه پروژه</b>
۳	۱.۱	چکیده
۳	۲.۱	مقدمه
۵	۳.۱	چالش ها
۶	۴.۱	کاربردها
۷	۲	<b>تحقیقات مقدماتی</b>
۷	۱.۲	تحقیق درباره نوع قطعات بورد اصلی و بورد ریموت کنترلر
۷	۱.۱.۲	برد اصلی پهپاد
۱۱	۲.۱.۲	لینک های مفید
۱۲	۲.۲	تحقیق راجع به نسخه اندروید مورد استفاده در ثابت افزار
۱۲	۳.۲	تحقیق راجع به فایل سیستم اندروید
۱۳	۴.۲	بررسی و جستجوی انجمن ها، فروم ها، و ریپازیتوری مرتبط با پهپادهای DJI و بررسی ابزارهای مختلف
۱۴	۳	<b>شبیه ساز</b>
۱۴	۱.۳	تحقیق در مورد شبیه سازهای پردازنده ARM
۱۴	۲.۳	بررسی و استفاده از QEMU برای شبیه سازی سیستم عامل های مختلف
۱۵	۱.۲.۳	نصب و اجرای QEMU
۲۱	۲.۲.۳	آزمایش ۱: شبیه سازی اندروید 8.1 نسخه پردازنده x86
۲۳	۳.۲.۳	آزمایش ۲: شبیه سازی rpios lite نسخه پردازنده ARM
۲۵	۴.۲.۳	آزمایش ۳: شبیه سازی اندروید 5.0 نسخه پردازنده arm
۲۶	۵.۲.۳	آزمایش ۴: شبیه سازی اندروید 4.4 نسخه پردازنده x86
۲۷	۴	<b>اجرا</b>
۲۷	۱.۴	دیکامپایل کردن برنامه موجود در ثابت افزار و بدست آوردن کد اندروید
۲۷	۱.۱.۴	تلاش یکم: Jad, fast and with Android support, online
۲۷	۲.۱.۴	تلاش دوم: Procyon - fast decompiler for modern Java
۲۷	۳.۱.۴	تلاش سوم: JDCore (very fast)
۲۸	۴.۱.۴	تلاش چهارم: CFR - very good and well-supported decompiler
۲۸	for modern Java	
۲۸	۵.۱.۴	تلاش پنجم: JAD (very fast, but outdated)
۲۸	۶.۱.۴	تلاش ششم (موفق): Jad, fast and with Android support on local machine
۲۹	۲.۴	تحلیل و تشریح کد اندروید بدست آمده از ثابت افزار
۲۹	۱.۲.۴	ابهام سازی یا Obfuscation
۳۰	۲.۲.۴	توابع دیکامپایل شده در تلاش ششم

۳۱ . . . . .	۳.۴ تحلیل firmware با ابزار firmwalk
۳۳ . . . . .	۴.۴ اجرای ثابت افزار مربوط به کنترلر بر روی QEMU
۳۳ . . . . .	۱.۴.۴ ساختار کلی فایل نصی اندروید
۳۴ . . . . .	۲.۴.۴ نحوه ایجاد فایل system.sfs
۳۹ . . . . .	۳.۴.۴ آزمایش ۱: شبیه سازی اندروید ۴.۴ نسخه پردازنده x86
۴۰ . . . . .	۴.۴.۴ آزمایش ۲: شبیه سازی نسخه دستکاری شده اندروید ۴.۴ با استفاده از فایل های دامپ
۴۱ . . . . .	۵.۴.۴ آزمایش ۳: شبیه سازی نسخه دستکاری شده اندروید ۴.۴ با استفاده از ترکیب فایل های دامپ و اندروید

## ۵ نتیجه گیری

## ۱ مقدمه پروژه

### ۱.۱ چکیده

در این پروژه قصد داریم تا ثابت افزار (Firmware) DJI Mavic 2 پهپاد را بررسی و تحلیل نماییم. روند کلی کار به این صورت است که در ابتدا این ثابت افزار بر روی شبیه ساز پردازنده ARM اجرا می شود و سپس اپلیکیشن هایی که در آن تعییه شده را به طور دقیق بررسی شده، و هدف، نحوه کار کرد، سورس کد، روتین ها و ماژول های این اپلیکیشن ها را استخراج می گردد.

در این گزارش در ابتدا اطلاعات کلی در مورد خود پهپاد و کنترلر و کاربردهای آن و همچنین سخت افزار استفاده شده بحث می شود. سپس در بخش دوم درباره QEMU و نحوه اجرای ثابت افزار بحث می شود و همچنین گزارش تعدادی آزمایش برای آشنایی با نحوه استفاده از QEMU قرار داده شده است.

### ۲.۱ مقدمه

با توجه به پیشرفت های قابل توجه تکنولوژی در سال های اخیر مخصوصا در زمینه های تصویر برداری، پروتکل های ارتباطی، سیستم های نهفته، و ارتباط از راه دور، پهپادها نقش قابل توجهی در عرصه های مختلف (عکس برداری هوایی، مقاصد نظامی و ...) یافته اند. به طوری که امروزه به پهپادها به چشم ابزارهای پیچیده جمع آوری اطلاعات در عرصه های مختلف نگاه می شود که می توانند یکی از ماده های اولیه نیاز اساسی بشر امروزی، یعنی داده خام را جمع آوری کنند. با استفاده از این داده های خام، می توان اطلاعات (information) و دانش (knowledge) استخراج کرد و گام های مهمی در جهت پیشرفت کسب و کار هدف برداشت.

امروزه از پهپاد استفاده های بسیاری می شود. به طور کلی به برخی از کاربردهای پهپادهای تجاری (commercial drones) در حوزه های مختلف می پردازیم:

- محافظت از طبیعت:** داده های جمع آوری شده توسط پهپادها از مناطق طبیعی می توانند به متخصصان طبیعت در جهت حفاظت از منطقه مدنظر کمک کند. برای مثال، یک تیم تحقیقاتی محافظت از اقیانوس ها از پلتفرم DJI برای جمع آوری داده برای مانیتور کردن وضعیت سلامت اقیانوس ها استفاده می کند. استفاده از این پلتفرم منجر به جمع آوری سریع، دقیق و ارزان داده می شود. [منبع]

- صنعت کشاورزی:** با استفاده از پهپادها و دوربین های مخصوص، می توان زمین های کشاورزی را به صورت بهتر و دقیق تر مانیتور و بررسی کرد و یا می توان از پهپاد جهت سمپاشی استفاده کرد. کاربردهای پهپادها در این زمینه به گونه ای زیاد است که شرکت مطرح DJI یک پلتفرم اختصاصی برای این حوزه توسعه داده است.

- تامین امنیت:** از پهپادها می توان جهت پایش و مانیتور کردن مکان های مهم (مانند مکان های نظامی) استفاده کرد و در صورت به تشخیص ناهنجاری (با استفاده از تکنیک های anomaly detection)، اقدامات لازم انجام شود. مزیت پهپادها نسبت به دوربین های مداربسته، قابلیت

جابجایی بیشتر آن‌ها و همچنین دوربین‌های قوی‌تر می‌باشد. پهپاد [DJI Matrice 300 RTK](#) برای این منظور ساخته شده است.

- **صنعت معدن:** از پهپادهای مخصوص برای بررسی ساختار داخلی معادن و تونل‌ها استفاده می‌شود. این نوع پهپادها دارای محافظ مخصوص می‌باشند و دارای حسگرهای منحصر به فرد جهت پیدا کردن در زمان قطع ارتباط، هستند. همچنین این نوع پهپادها دارای تجهیزات ارتباطی بسیار قوی می‌باشند و می‌توانند از اعمق زمین نیز ارتباط با مرکز کنترل را حفظ کنند. پهپاد [Elios 2](#) از این نوع می‌باشد.
- **فیلمبرداری و تصویربرداری:** می‌توان گفت که پرکاربردترین کارکرد پهپادهای تجاری، فیلمبرداری و تصویربرداری است. با استفاده از پهپادها می‌توان تصاویر خیره کننده‌ای را ثبت کرد. همچنین امروزه در بخش فیلمبرداری اکثر فیلم‌های سینمایی بزرگ، از پهپادها استفاده می‌گردد. سری [Mavic](#) شرکت [DJI](#) یکی از بهترین پهپادها برای این کارکرد می‌باشد.
- **نقشه‌برداری:** با استفاده از نوع خاصی از پهپاد، می‌توان از یک منطقه جغرافیایی، نقشه‌برداری کرد. این پهپاد یک سری عکس از ارتفاع مخصوص از سطح زمین تهیه می‌کنند و سپس با استفاده از یک سیستم هوشمند و ایجاد یک مدل سه بعدی، به نقشه‌برداری می‌پردازند. پهپاد [WingtraOne](#) یکی از بهترین پهپادها در این زمینه است.
- **جمع‌آوری داده:** با توجه به مجهز بودن اکثر پهپادها به دوربین با کیفیت و سایز کوچک پهپاد، از آن‌ها برای جمع‌آوری تصاویر استفاده می‌شود. با توجه به نیاز روزافزون تکنولوژی‌های جدید مانند هوش مصنوعی و یادگیری عمیق به داده بسیار زیاد و با کیفیت، پهپادها می‌توانند نقش قابل توجهی در این زمینه داشته باشند.

در سال‌های اخیر شرکت‌های مختلفی با هدف تولید پهپاد تجاری برای مقاصد مختلف بوجود آمده است که موارد زیر جزو مطرح‌ترین آن‌ها می‌باشند:

- DJI
- Parrot
- Yuneec
- Kespry
- Autel Robotics
- Skydio
- Insitu
- Delair
- Ehang

- Aerialtronics

مانند تمام ابزارهای جدید، پهپادهای تجاری نیز دارای محدودیت‌های مخصوص خود می‌باشند. این محدودیت‌ها معمولاً در بخش نرم افزار (به طور دقیقتر ثابت افزار یا firmware) پهپادها می‌باشد. برای مثال محدودیتی به نام (No Fly Zones (NFZ) در این نوع پهپادها وجود دارد که مانع از پرواز آن‌ها در مناطق محافظت شده (مانند حریم هوایی فرودگاه‌ها) می‌شود. البته برخی از این محدودیت‌ها لازم و ضروری هستند اما تعدادی از این موارد، محدودیت‌هایی هستند که مانع از عملکرد کامل و دقیق پهپاد شود. درنتیجه مباحث سیستمی، نرم افزاری، سخت افزاری و تکنیک‌ها هک و مهندسی معکوس، برای ایجاد تغییرات و برداشتن این نوع محدودیت‌ها استفاده می‌گردد.

همچنین با توجه به وجود انجمن (forum) و community‌های مختلف در سطح اینترنت و فعالیت عده کثیری از کاربران حرفه‌ای در این زمینه، آموزش‌های بسیاری در زمینه نحوه هک و دور زدن محدودیت‌ها در سطح اینترنت وجود دارد. همچنین تعدادی نرم افزار اختصاصی نیز برای این منظور تهیه شده است. انجمن‌ها، منابع و نرم افزارهای مخصوص این کار (مخصوص پهپادهای DJI) به صورت زیر می‌باشند:

- [فروم رسمی DJI](#)
- [فروم MavicPilots](#)
- [فروم phantomhelp](#)
- [ریپازیتوری dji-firmware-tools](#)
- [ریپازیتوری DUMLDore](#)

در این پروژه قصد داریم تا با بررسی کامل [ثابت افزار](#) کنترلر پهپاد 2 DJI Mavic و اجرای آن بر روی شبیه‌ساز پردازنده ARM، به تحلیل و آنالیز آن بپردازیم. هدف از این تحلیل بررسی کامل قابلیت‌ها و اپلیکیشن‌های موجود بر روی کنترلر پهپاد می‌باشد. با انجام این تحلیل، می‌توان تغییراتی را در کنترلر به وجود آورد و یا حتی می‌توان با استفاده از روش‌های مهندسی معکوس توانایی تولید کنترلر مشابه را بدست آورده.

همچنین علاوه بر تلاش برای اجرای ثابت افزار بر روی شبیه‌ساز، بخش‌های مختلف فایل دامپ ثابت افزار بررسی شد و با استفاده از ابزارهای مختلف (مانند binwalk و firmwalk) ابعاد دیگری از ثابت افزار مورد بررسی قرار گرفت. همچنین فایل jar موجود بر در یکی از دایرکتوری‌های ثابت افزار بررسی و دیکامپایل گردید.

## ۳۰.۱ چالش‌ها

با توجه به ماهیت پروژه طبیعتاً در اجرای پروژه، وجود چالش‌ها و سختی‌های مختلف اجتناب‌ناپذیر است. یکی از چالش‌هایی که در اجرای پروژه به آن برخوردهایم، موجود نبودن اطلاعات کافی و دقیق در مورد خود سخت افزار و همچنین نحوه اجرای ثابت افزار روی سخت افزار مشابه بود. با توجه به تحقیقات و جستجوهایی که انجام شد، اکثر بحث‌های گروه‌ها و کامیونیتی‌ها حول محور ایجاد تغییر در ثابت افزار صرفاً

با استفاده از یک سری روش‌های بسیار ساده بود و بحثی در مورد نحوه اجرا و تحلیل ثابت افزار یافته نشد. البته تعدادی سؤال در این فرومها در مورد ابهامات و نحوه ایجاد شبیه‌ساز از چند نفر از افراد با تجربه پرسیده شد اما تاکنون پاسخ خاصی دریافت نشده است. همچنین یکی از پیش‌نیازهای اعمال برخی از تغییرات در ثابت افزار پهپاد، داشتن خود پهپاد بود و بدون داشتن پهپاد نمی‌توان به هک برخی از قسمت‌های آن پرداخت.

## ۴.۱ کاربردها

نتیجه این پژوهه کاربردهای مختلفی می‌تواند داشته باشد. برای مثال با استفاده از روش‌های مهندسی معکوس می‌توان نحوه ارتباط میان کنترلر و پهپاد را بدست آورد و همچنین ساختار کلی کنترلر را نیز کشف کرد. پس از آن نیز می‌توان کنترلر شخصی سازی شده برای کنترل کردن پهپاد ایجاد کرد.

## ۲ تحقیقات مقدماتی

### ۱.۲ تحقیق درباره نوع قطعات بورد اصلی و بورد ریموت کنترلر

یک نمونه از محصول مورد بررسی در اختیارمان بود اما باز کردن آن به دلایل مختلف امکان پذیر نبود. برای مثال مطمئن نیستیم که آیا باز کردن دستگاه موجب فعال شدن قابلیت خودتخریبی (در صورت وجود) می شود یا خیر. بنابراین به جستجو در فضای نت پرداختیم. مدل دستگاه را جستجو کرده و تصاویر و بلاگ های مربوطه را مطالعه کردیم. در نتیجه این جستجوها به چند تصویر که مدل و نوع ماژول های روی بورد در آن ها مشخص است و یک مخزن گیت هاب رسیدیم که تا حدودی به توضیح و تشریح ماژول های مورد استفاده در این محصول پرداخته است. متاسفانه تصویر قابل استفاده ای از بورد ریموت کنترلر نیافتنیم و همچنین مخزن مذکور اطلاعاتش تکمیل نیست.

#### ۱.۰۲ برد اصلی پهپاد

برد اصلی پهپاد 2 pro mavic که با نام [WM240](#) شناخته می شود، وظیفه پردازش، ذخیره سازی و ارسال اطلاعات را دارد. این برد شامل بخش های مختلفی از جمله video encoder (هم برای FPV و هم برای SD-Card VPS)، gesture recognition for positioning (یا gesture)، (یا Transceiver) برای ارتباطات رادیویی می باشد. همچنین برد دارای فرستنده و گیرنده (Transceiver) برای ارتباطات رادیویی می باشد.

اجزای اصلی برد پهپاد موارد زیر می باشند.

- **H3**

پردازنده [Allwinner H3 \(sun8iw7p1\)](#) وظایف انکود کردن ویدیو، پردازش مباحث vision و پهپاد و کنترل پرواز را بر عهده دارد.

- **LC1860C**

پردازنده [Leadcore LC1860C](#) وظایف ارتباطات رادیویی و کنترل پرواز هوشمند را بر عهده دارد.

- **آی سی**

از آی سی برای تامین ولتاژ و جریان در مسیر اجرای دستور، و همچنین برای حفاظت در مدار و تامین انرژی آی سی ها استفاده می شود. از [Leadcore LC1160](#) به عنوان آی سی در برد پهپاد استفاده می شود.

- **RAM**

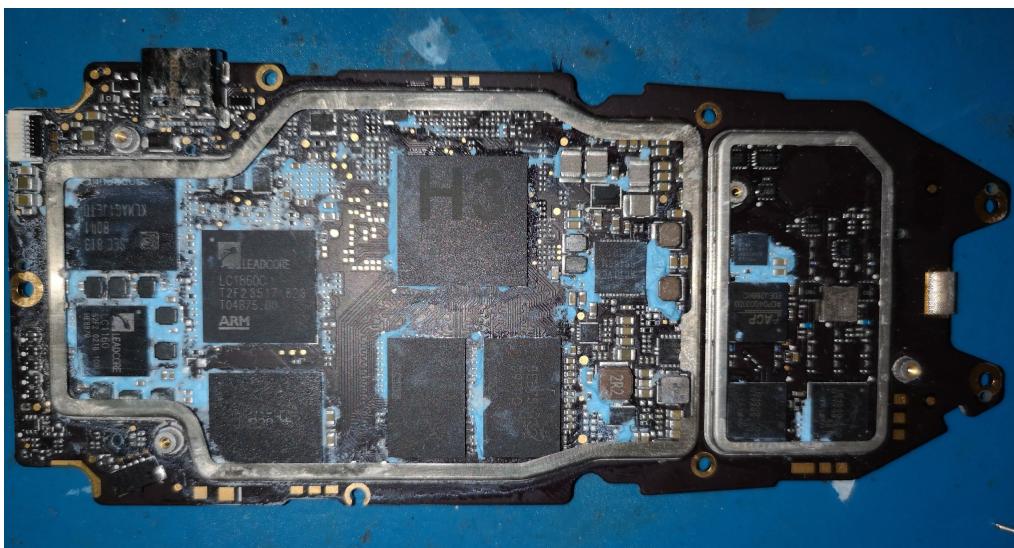
طبق بررسی، از ماژول [Micron MT29TZZZ4D4BKRL - ۱۲۵](#) به عنوان حافظه اصلی استفاده می شود. ، این برد دارای ۸ گیگابایت DDR4 SDRAM می باشد.

- **حافظه**

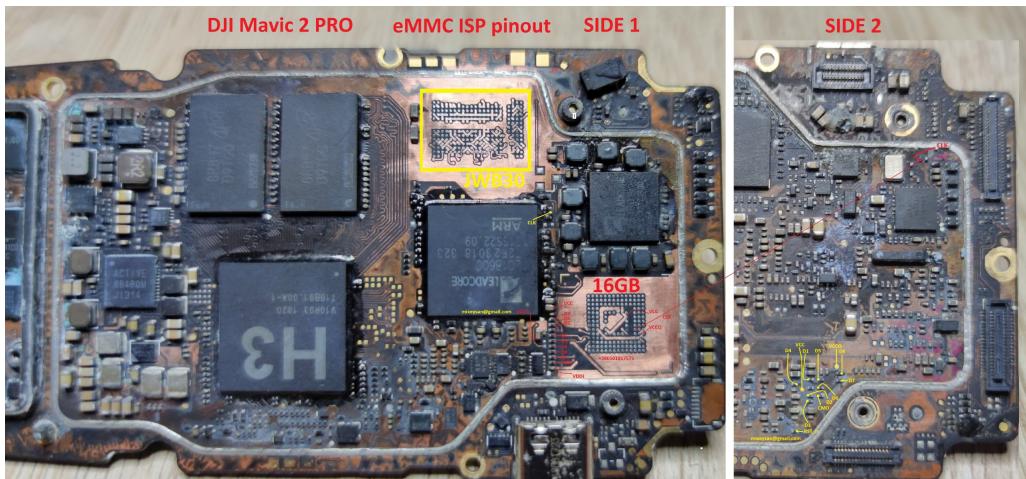
از ماژول Samsung KLMAG1JETD-B041 به عنوان حافظه جانبی پهپاد استفاده می شود. البته لازم به ذکر است که می توان از SD Card نیز در پهپاد استفاده کرد.



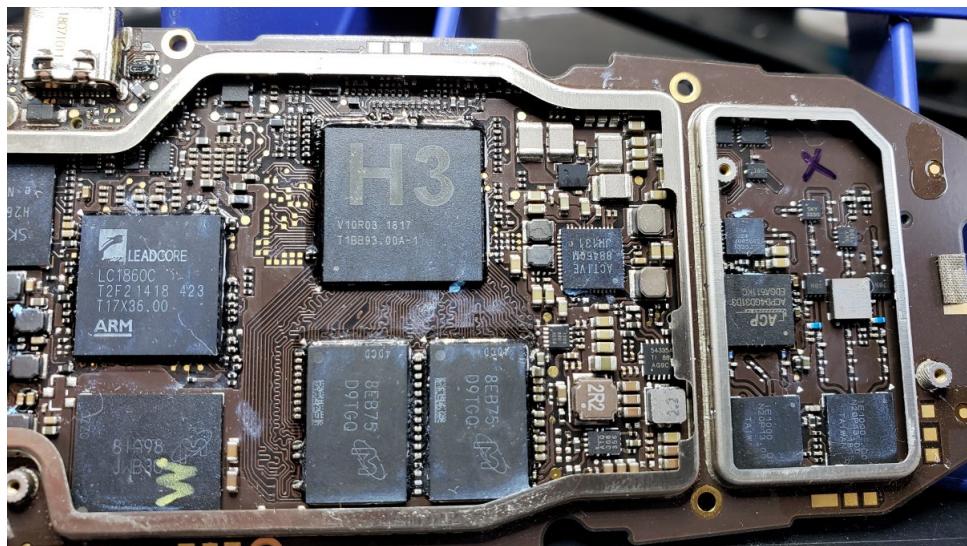
شکل ۱: برد اصلی پهپاد پیش از جداسازی محافظ فلزی - منبع



شکل ۲: برد اصلی پهپاد پس از جداسازی محافظ فلزی - منبع



شکل ۳: پشت و روی بورد اصلی پهبد



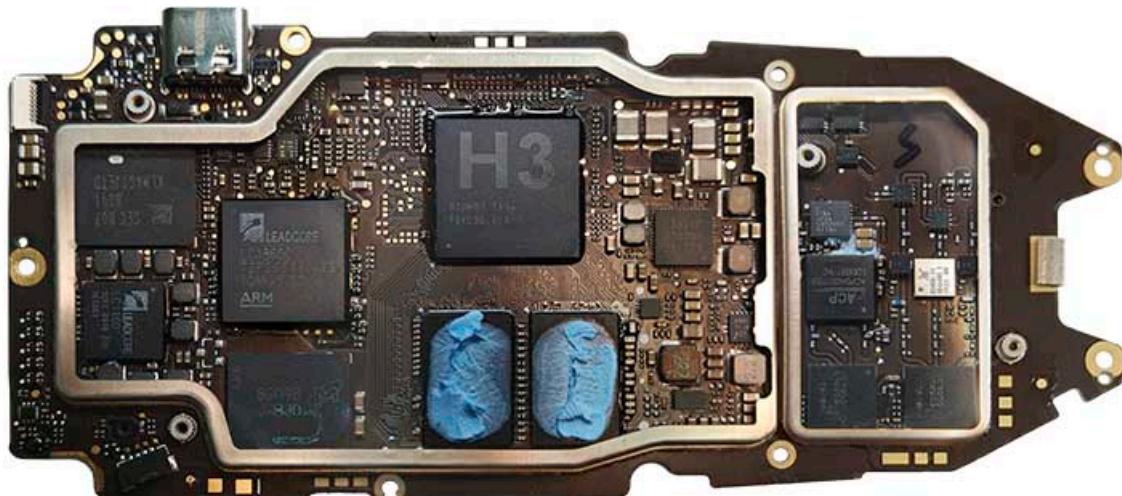
شکل ۴: تصویری واضح از بورد اصلی پهبد و قطعات و مازولهای مورد استفاده – منبع

**Active-Semi Advanced PMIC •**

مدار مجتمع مدیریت نیرو ACTIVE 8846QM ([PMIC](#)) برای مدیریت اپلیکیشن های چند هسته ای پهپاد استفاده می گردد.

**مودم •**

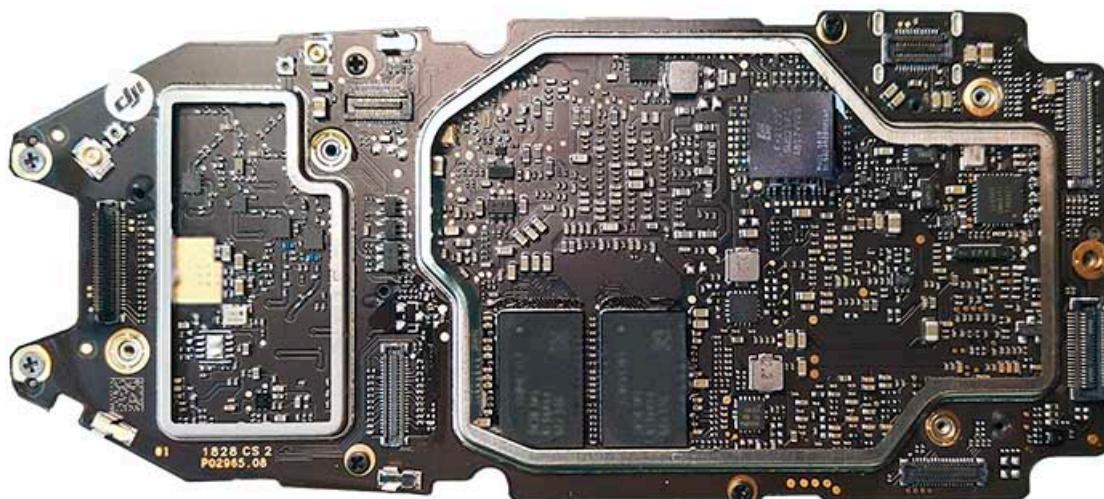
از مودم LTE با نام ACP IRIS411 در پهپاد استفاده می شود.

**ماژول GPS •****بورد سنسور دوربین •****باتری •**

شکل ۵: تصویر جلوی بورد اصلی پهپاد



شکل ۶: پشت برد اصلی پهیاد پیش از جداسازی محافظ فلزی



شکل ۷: پشت برد اصلی پهیاد پس از جداسازی محافظ فلزی

## ۲۰.۲ لینک‌های مفید

- [ریپاریتوری مرجع برای ابزارهای مرتبط با ثابت افزار DJI](#)

- صفحه اطلاعات برد اصلی پهپاد

mavic 2 prop

## ۲.۰۲ تحقیق راجع به نسخه اندروید مورد استفاده در ثابت افزار

به جهت راه اندازی سامانه بر روی شبیه ساز به سراغ یافتن نسخه اندروید مورد استفاده از کوادکوپتر رفتیم. برای این منظور ابتدا با جستجو در فضای اینترنت به نتیجه هی مشخصی نرسیدیم اما سپس با بررسی فایل های موجود در فایل سیستم، به سندی تحت عنوان build.prop بخوردیم که اطلاعاتی راجع به کوادکوپتر در داخلش نوشته شده است. این فایل در عموم سیستم های اندرویدی وجود دارد و به کمک آن می توان ویژگی هایی از سیستم را تنظیم کرد. پارامترهای مهم موجود در این فایل در ثابت افزار مورد بررسی ما موارد زیر هستند:

```

¹ ro.build.display.id=leadcore1860
² ro.build.version.sdk=19
³ ro.build.version.release=4.4.4
⁴ ro.product.model=L1860
⁵ ro.product.brand=Leadcore
⁶ ro.product.name=full_wm240_dz_rp0010_v1
⁷ ro.product.cpu.abi=armeabi-v7a
⁸ net.bt.name=Android

```

همانطور که پیداست، از اندروید نسخه ۴.۴.۴ استفاده شده است.

## ۳.۰۲ تحقیق راجع به فایل سیستم اندروید

از آنجا که اندروید فریم ورکی<sup>۱</sup> برای لینوکس است، ساختار فایل سیستم<sup>۲</sup> آن هم شباهت زیادی به فایل سیستم لینوکس دارد. با این حال فایل سیستم اندروید ویژگی هایی مختص به خود و متفاوت با لینوکس را هم دارد.

در فایل سیستم اندروید به طول معمول ۶ پارتیشن اصلی وجود دارد.

- \\boot
- \\system
- \\recovery
- \\data

---

Framework<sup>۱</sup>  
File system<sup>۲</sup>

- \cache
- \misc
- { \sdcard }
- { \sd-ext }

## ٤٠٢ بررسی و جستجوی انجمن‌ها، فروم‌ها، و ریپازیتوری مرتبط با پهپادهای DJI و بررسی ابزارهای مختلف

جهت یافتن اطلاعات بیشتر در مورد نحوه اجرای پروژه و همچنین اطلاعات دقیق در مورد سخت افزار پروژه، سایتها، فرومها و ریپازیتوری‌های مختلف بسیاری در مورد پهپادهای DJI بررسی شد. برای نمونه فروم‌های اصلی DJI و mavicpilots بررسی شد و همچنین یک سری سؤال در مورد نحوه اجرای ثابت افزار پرسیده شد. همچنین چندین ریپازیتوری مختلف در سایت گیت‌هاب موردنبررسی قرار گرفت. در ادامه لینک چندین منبع و بحث در کامیونیتی‌های مختلف در مورد این موضوع قرار داده شده است:

- بحث در مورد نحوه هک کردن ویژگی‌های مختلف پهپاد Mavic 2
- بحث در مورد نحوه هک کردن مجوز FCC پهپادهای Mavic 2
- پرسش در مورد نحوه اجرای ثابت افزار بر روی شبیه‌ساز
- نکات مرتبط با پهپادهای Mavic
- صفحه اصلی روش‌های هک پهپادهای DJI
- ریپازیتوری DUMLDore برای هک و ایجاد تغیرات در پهپاد
- ریپازیتوری DJI Firmware Tools، منبع جامع و کامل برای ابزارهای مدیریت ثابت افزار DJI

### ۳ شبیه ساز

#### ۱.۰۳ تحقیق در مورد شبیه ساز های پردازنده ARM

به منظور اجرا و بازسازی کردن شرایط واقعی اجرای برنامه نیازمند یک شبیه ساز هستیم که با توجه به بورد و قطعات مورد استفاده در پهاد نیازمند یک شبیه ساز پردازنده ARM هستیم.

شبیه ساز های نرم افزاری ARM به شما این امکان را می دهند که یک دستگاه ARM شبیه سازی شده را بر روی سیستم اصلی کامپیوتر خود اجرا کنید، خواه ویندوز، لینوکس یا هر سیستم عامل دیگری باشد. این به شما امکان توسعه و آزمایش نرم افزار را بر روی سیستم شخصی خودتان می دهد و می توانید در زمان نیاز، آن را به یک دستگاه واقعی ARM منتقل کنید.

در صورت ناموفق بودن این تلاش، اقدام بعدی می تواند خرید قطعات اصلی و اجرای کد مستقیما بر روی قطعات فیزیکی باشد. مشخص است که این گزینه هزینه بیشتری را در بر دارد.  
به منظور انتخاب شبیه ساز مناسب چندین گزینه پیش رو داریم.

- ARMware
- Microsoft Device Emulator 3.0 (Standalone Release)
- Softgun - the Software ARM
- QEMU CPU Emulator
- SkyEye

از میان این شبیه سازها بهترین انتخاب از نظر سادگی در استفاده و جامع بودن شبیه ساز QEMU است. چرا که سایر شبیه سازها از طیف گسترده ای از پردازنده های پشتیبانی نمی کنند و مستندات و آموزش های موجود از آن ها هم به اندازه QEMU جامع و کامل نیست. بنابر دلایل بالا ما در انجام پروژه خود از شبیه ساز QEMU استفاده کردیم.

#### • معرفی شبیه ساز های آزاد ARM

#### ۲.۰۳ بررسی و استفاده از QEMU برای شبیه سازی سیستم عامل های مختلف

برای آشنایی بیشتر با شبیه ساز QEMU و بررسی نحوه ایجاد محیط مجازی، چندین آزمایش کلی در این زمینه انجام شد، که نحوه اجرای آن ها در این بخش توضیح داده می شوند.

### ۱۰.۳ نصب و اجرای QEMU

در این بخش قصد داریم تا با نحوه نصب QEMU و بخش های مختلف آن آشنا شویم. در ابتدا به نحوه نصب این شبیه ساز بر روی Ubuntu 20.04 می پردازیم. با اجرای دو دستور زیر، خود QEMU بهمراه نیازمندی ها نصب می شوند:

```
1 $ sudo apt update
2 $ sudo apt install qemu-kvm virtinst qemu virt-manager libvirt-daemon \
3 libvirt-daemon-system bridge-utils virt-viewer libvirt-clients
```

علاوه بر qemu، مازول qemu-kvm (برای استفاده از kvm در qemu جهت افزایش سرعت اجرا)، مازول UI virt-manager (یک UI مبتنی بر qemu برای اجرا و کانفیگ شبیه سازی)، و libvirt (شامل فایل های موردنیاز qemu جهت شبیه سازی) نصب می گردد. با استفاده از دستور زیر نیز، نصب فایل های موردنیاز بررسی می گردد.

```
1 $ sudo systemctl status libvирtd
```

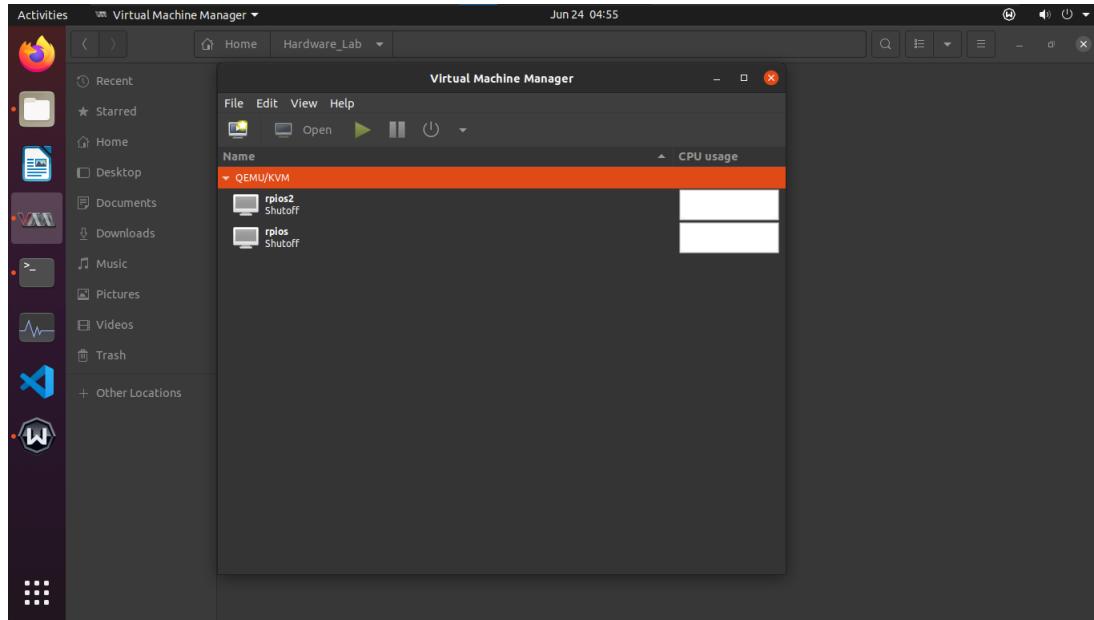
```
libvирtd.service - Virtualization daemon
   Loaded: loaded (/lib/systemd/system/libvирtd.service; enabled; vendor preset: enabled)
   Active: active (running) since Mon 2022-06-13 11:36:01 PDT; 1 weeks 3 days ago
     TriggeredBy: ● libvирtd.socket
                  ● libvирtd-ro.socket
                  ● libvирtd-admin.socket
     Docs: man:libvирtd(8)
           https://libvирtd.org
    Main PID: 965 (libvирtd)
      Tasks: 19 (limit: 32768)
     Memory: 2.4M
        CGroup: /system.slice/libvирtd.service
                 └─ 965 /usr/sbin/libvирtd
                     ├─1297 /usr/sbin/dnsmasq --conf-file=/var/lib/libvирtd/dnsmasq/default.conf --leasefile-ro --dhcp-script=/usr/lib/libvирtd/libvирtd
                     ├─1298 /usr/sbin/dnsmasq --conf-file=/var/lib/libvирtd/dnsmasq/default.conf --leasefile-ro --dhcp-script=/usr/lib/libvирtd/libvирtd

Jun 16 13:43:57 ubuntu dnsmasq[1297]: reading /etc/resolv.conf
Jun 16 13:43:57 ubuntu dnsmasq[1297]: using nameserver 127.0.0.53#53
Jun 16 13:43:59 ubuntu dnsmasq[1297]: reading /etc/resolv.conf
Jun 16 13:43:59 ubuntu dnsmasq[1297]: using nameserver 127.0.0.53#53
Jun 16 13:43:59 ubuntu dnsmasq[1297]: reading /etc/resolv.conf
Jun 16 13:43:59 ubuntu dnsmasq[1297]: using nameserver 127.0.0.53#53
Jun 24 00:09:20 ubuntu dnsmasq[1297]: reading /etc/resolv.conf
Jun 24 00:09:20 ubuntu dnsmasq[1297]: using nameserver 127.0.0.53#53
Jun 24 00:09:20 ubuntu dnsmasq[1297]: reading /etc/resolv.conf
Jun 24 00:09:20 ubuntu dnsmasq[1297]: using nameserver 127.0.0.53#53
...
lines 1-26/26 (END)
```

شکل ۸: صحیح بودن نصب QEMU

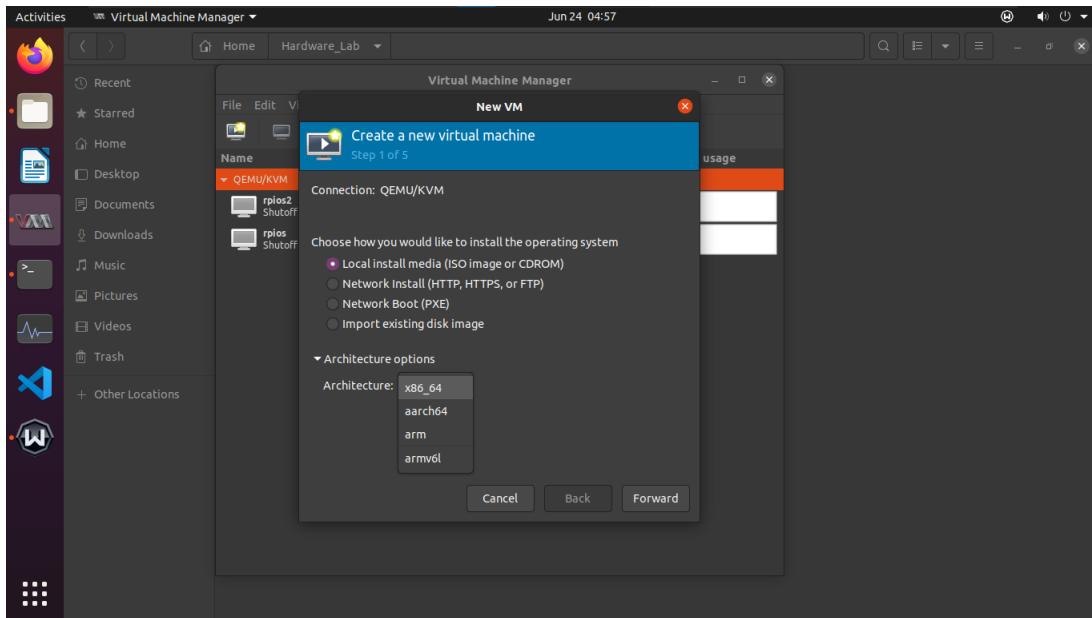
برای اجرای QEMU و شبیه سازی دو راه وجود دارد. یکی از این راهها استفاده از ترمینال لینوکس و اجرای کامند های مربوطه می باشد. روش دوم استفاده از virt-manager می باشد که به نوعی یک UI برای QEMU محسوب می گردد و می توان با استفاده از این ابزار، به راحتی کانفیگ های مربوطه را انجام داد و شبیه سازی را اجرا کرد. در آزمایش های انجام شده از هر دو روش استفاده شده است؛ اما برای راحتی، صرفا

کامندهای مربوطه به ازای هر آزمایش نوشته شده است. برای آشنایی بیشتر، به اجرای یک شبیه‌سازی با استفاده از virt-manger می‌پردازیم.



شکل ۹ : تصویری از محیط اجرای virt-manager

در ابتدا گرینه machine virtual new a Create را انتخاب می‌کنیم.

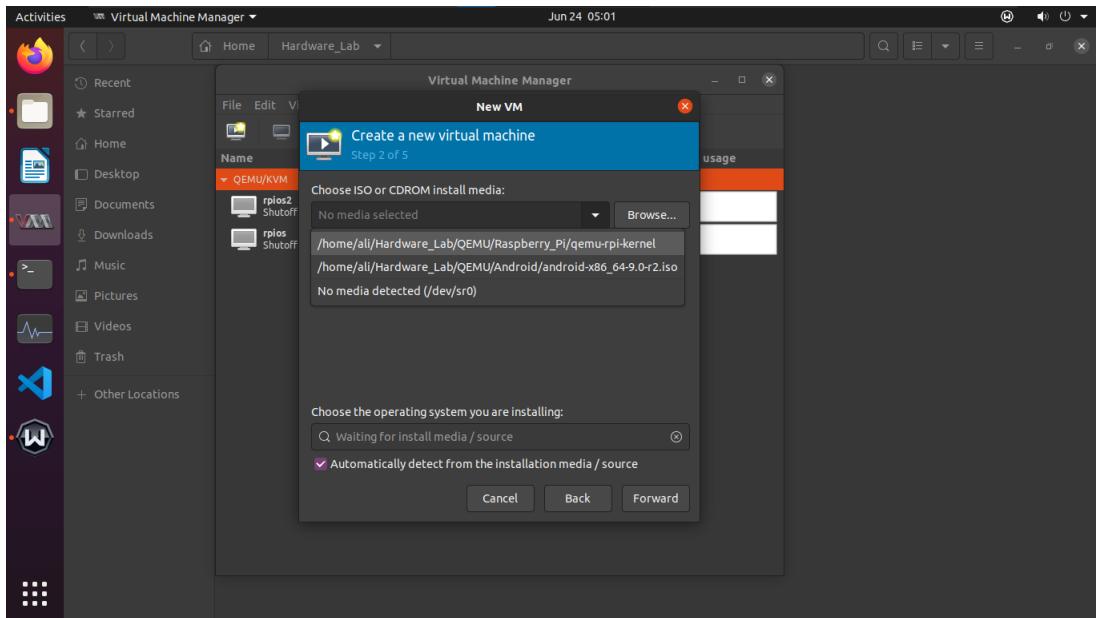


شکل ۱۰ : گزینه های مختلف برای ایجاد یک VM

در پنجره جدید می توان معماری مورد استفاده پردازنده شبیه ساز را نیز انتخاب کرد. QEMU از طیف وسیعی از پردازنده های مختلف با معماری های ARM و X86 پشتیبانی می کند. برای آشنایی بیشتر با این پردازنده ها می توانید از دو لینک زیر استفاده کنید:

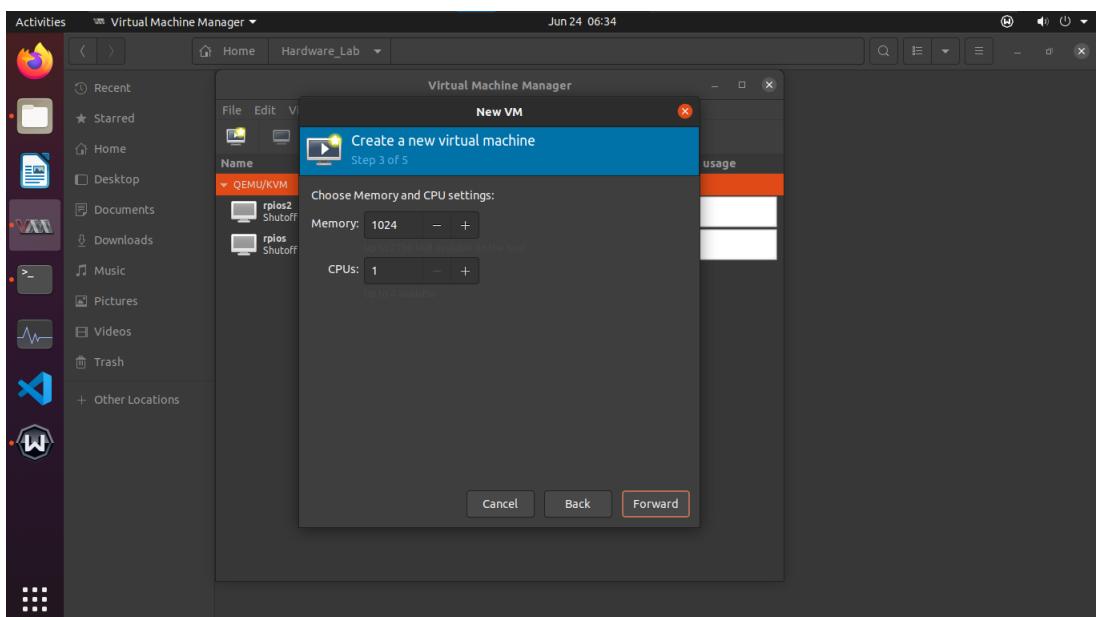
- [لینک ۱](#)
- [لینک ۲](#)

صرفاً گزینه های پیش فرض را انتخاب می کنیم و گزینه Forward را می زنیم.

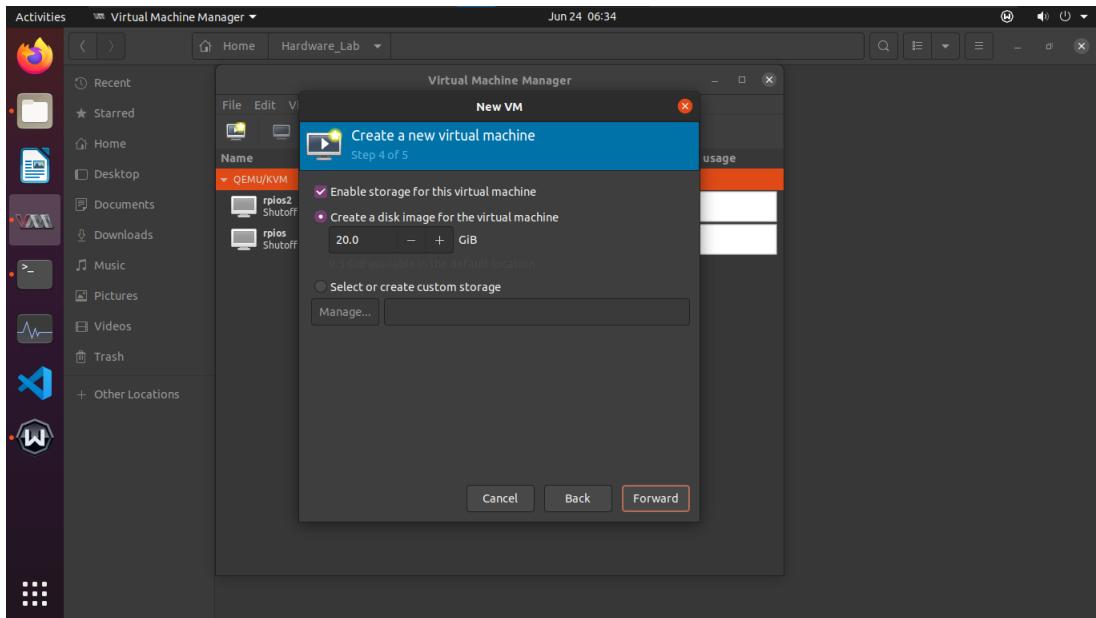


شکل ۱۱ : بخش انتخاب فایل iso

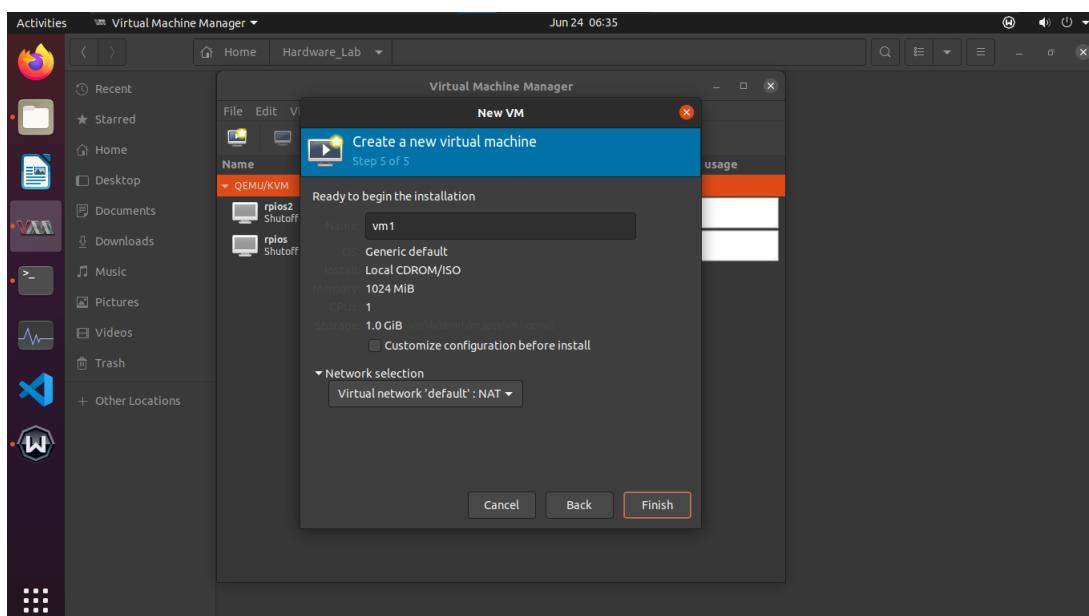
سپس باید فایل نصبی سیستم عامل را انتخاب کرده و سیستم عامل مورد نظر را از طریق گزینه های موجود انتخاب نماییم. صرفا برای آزمایش Generic default را به عنوان سیستم عامل انتخاب می نماییم.



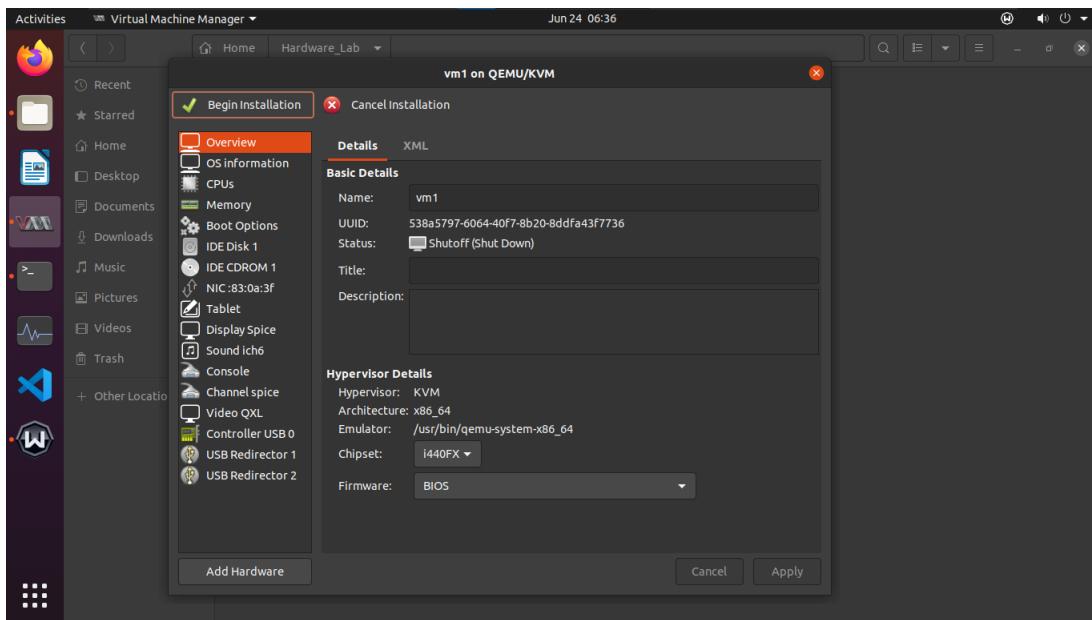
شکل ۱۲ : انتخاب مشخصات سیستم



شکل ۱۳: انتخاب مشخصات حافظه

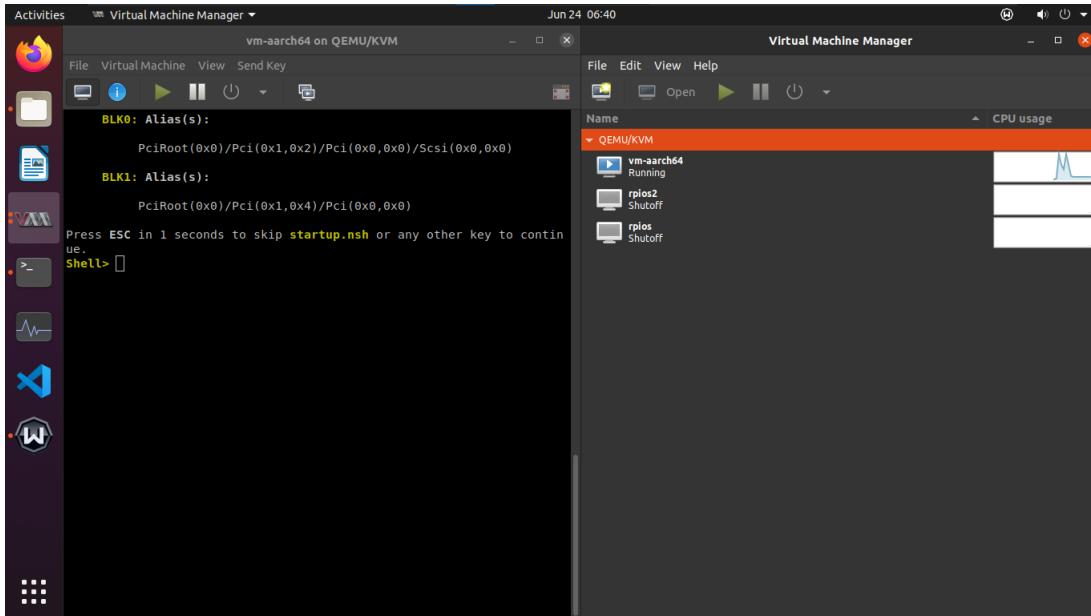


شکل ۱۴: نهایی کردن مشخصات و اجرا



شکل ۱۵ : نهایی کردن مشخصات و اجرا

لازم به ذکر است که با انتخاب گزینه Customize configuration before install می‌توان مشخصات سیستم را با جزیيات بیشتری تغییر داد و یا ابزارها و سخت افزارهای دیگری نیز به آن افزود. پس از تنظیم مشخصات سیستم، به اجرای شبیه ساز می‌پردازیم.



شکل ۱۶: اجرای شبیه‌ساز

با توجه به اینکه هیچ‌گونه فایل نصی مرتبط با سیستم عامل به عنوان ورودی به QEMU داده نشده است، طبیعتاً هیچ سیستم عامل خاصی شبیه‌سازی نمی‌شود و صرفاً کامندلاین پیش‌فرض QEMU اجرا می‌گردد. برای آشنایی بیشتر با این شبیه‌ساز می‌توان از [دکیومنت‌های](#) غنی آن نیز بهره برد. در بخش‌های بعدی به اجرای سیستم عامل‌های مختلف بر اساس معماری‌های مختلف و با استفاده از کامندلاین اوینتو می‌پردازیم. لازم به ذکر است که attribute های موجود در هر کامند در بخش‌های بعدی، منتظر با مشخصات گفته شده در virt-manager می‌باشند و می‌توان هر کدام از آن‌ها را در بعدی، پیدا کرد و تغییرات را اعمال نمود.

### ۲۰۲۳ آزمایش ۱: شبیه‌سازی اندروید ۸.۱ نسخه پردازنده x86

در ابتدا لازم است که فایل نصی اندروید ۱.۸ دانلود شود. با استفاده از این [لینک](#) می‌توان این کار را انجام داد.

در ابتدا لازم است که با استفاده از کامند زیر، یک Drive Hard مجازی ایجاد شود:

```
$ qemu-img create -f qcow2 androidx86_hda.img 10G
```

سپس با استفاده از دستور زیر، فایل نصی را اجرا می‌کنیم.

```
$ qemu-system-x86_64 \
-m 2048 \
-smp 2 \
```

```

` -soundhw es1370 \
` -device virtio-mouse-pci -device virtio-keyboard-pci \
` -serial mon:stdio \
` -boot menu=on \
` -net nic \
` -net user,hostfwd=tcp::5555-:22 \
` -device virtio-vga,virgl=on \
` -display gtk,gl=on \
` -hda androidx86_hda.img \
` -cdrom android-x86_64-1.8-r3.iso

```

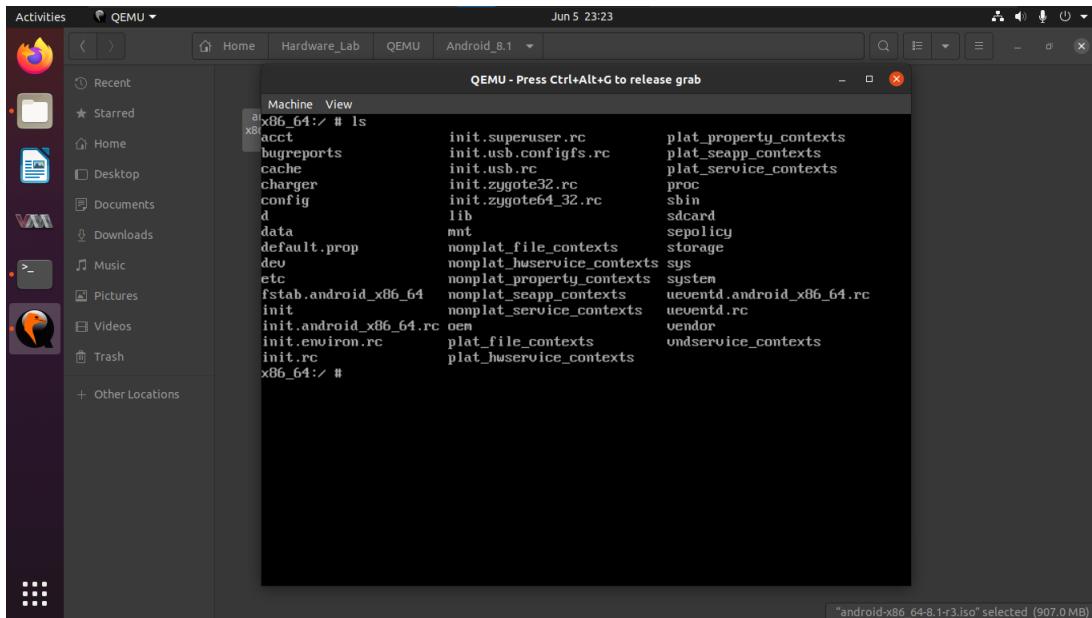
با استفاده از آموزش موجود در [لينك](#) به نصب اندروید می‌پردازیم. البته می‌توان به طور مستقیم و بدون نصب، به اجرای سیستم عامل پرداخت. بعد از نصب کامل سیستم عامل اندروید، با استفاده از کامند زیر می‌توان سیستم عامل را اجرا کرد.

```

` $ qemu-system-x86_64 \
` -m 2048 \
` -smp 2 \
` -soundhw es1370 \
` -device virtio-mouse-pci -device virtio-keyboard-pci \
` -serial mon:stdio \
` -boot menu=on \
` -net nic \
` -net user,hostfwd=tcp::5555-:22 \
` -device virtio-vga,virgl=on \
` -display gtk,gl=on \
` -hda androidx86_hda.img

```

در شکل زیر ترمینال اندروید 8.1 را مشاهده می‌نمایید. با توجه به اینکه هدف این آزمایش صرفاً آشنایی با ساختار QEMU و نحوه اجرای یک سیستم عامل با این شبیه‌ساز است، صرفاً اجرای اندروید بر روی ترمینال کافی بود.



شکل ۱۷: ترمینال اندروید ۸.۱ اجرا شده بر روی QEMU

### ۳.۰.۳ آزمایش ۲: شبیه‌سازی rpios lite نسخه پردازنده ARM

در آزمایش دوم به نصب lite نسخه مخصوص پردازنده‌های آرم پرداختیم. با فرض نصب بودن، QEMU در ابتدا لازم است که فایل سیستم عامل دانلود شود. با استفاده از دستور زیر به دانلود نسخه مشخصی از سیستم عامل rpi می‌پردازیم.

```

$ wget https://downloads.raspberrypi.org/rpios_lite_armhf/images/
rpios_lite_armhf-2021-01-12/2021-01-11-rpios-buster-armhf-lite.zip

```

پس از آن، لازم است که کرنل‌های مورد نیاز این سیستم عامل از طریق این [دیپاپزیتوری](#) دانلود شود.

```

$ git clone https://github.com/dhruvvyas90/qemu-rpi-kernel

```

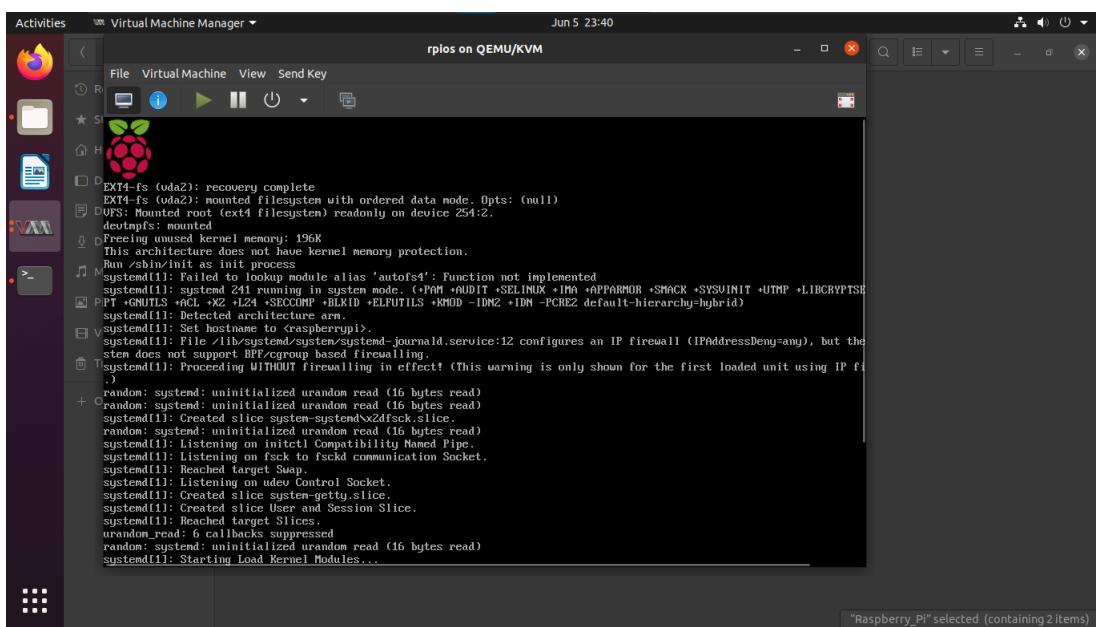
اکنون با استفاده از کامند زیر، سیستم عامل Lite Pi Raspberry را بالا اجرا می‌کنیم:

```

$ sudo virt-install \
--name rpios \
--arch armv6l \
--machine versatilepb \
--cpu arm1176 \
--vcpus 1 \
--memory 256 \

```

```
^ --import \
  --disk 2021-01-11-raspios-buster-armhf-lite.img,format=raw,bus=virtio \
. --network bridge,source=virbr0,model=virtio \
.. --video vga \
. --graphics spice \
.. --boot 'panic=1 ,kernel_args=root=/dev/vda2,kernel=qemu-rpi-kernel/kernel-qemu-4.50
. --events on_reboot=destroy
```



شکل ۱۸: لود شدن سیستم عامل os rpi بر روی QEMU

```

pi@raspberrypi: ~ ls
all
pi@raspberrypi: ~ $ cd ..
pi@raspberrypi: /home $ ls
pi
pi@raspberrypi: /home $ cd ..
pi@raspberrypi: ~ $ ls
bin boot dev etc home lib lost+found media mnt opt proc root run sbin srv sys tmp usr var
pi@raspberrypi: ~ $ uname -a
Linux raspberrypi 4.19.50+ #1 Tue Nov 26 01:49:16 CET 2019 armv6l GNU/Linux
pi@raspberrypi: ~ $

```

شکل ۱۹ : ترمینال rpi os اجرا شده بر روی QEMU

### ۴.۰.۳ آزمایش ۳: شبیه سازی اندروید ۵.۰ نسخه پردازنده arm

برای آشنایی بیشتر با اجرای سیستم عامل روی شبیه ساز QEMU و همچنین با توجه به این نکته که ثابت افزار مورد بررسی بر مبنای اندروید ۵.۰ نسخه arm می باشد، قصد داریم تا این نسخه از اندروید (متینی بر arm) را بر روی QEMU اجرا نماییم. در ابتدا برای اجرای این نسخه از اندروید، در مورد آن جستجو و تحقیق کردیم و به این پرسش در سایت StackOverFlow برخوردیم که در آن به این موضوع اشاره شده است که احتمالاً روش مناسبی برای انجام این کار وجود ندارد. همچنین با بررسی بیشتر دو مقاله در سطح اینترنت یافت شد که در مورد اجرای این کار توضیحات ارائه کرده بود. این مقالات دو اشکال اساسی داشتند؛ اشکال اول، قدیمی بودن این مقالات می باشد (مربوط به سال ۲۰۱۴) و در نتیجه در دسترس نبودن اکثر منابع مورد نیاز برای اجر می باشد. اشکال دوم مربوط به سخت افزار مورد نیاز برای شبیه سازی درست این نسخه از اندروید است. به طور مثال، در یکی از این مقالات، نویسنده از یک سیستم دارای پردازنده AMD FX-8350 با ۸ هسته پردازشی و فرکانس 4.2 گیگاهرتز به همراه 16 گیگ RAM استفاده کرده بود و به طولانی بودن اجرا بخشی از قسمت های شبیه سازی (در حد یک روز) اشاره کرده بود.

- مقاله Running Android L Developer Preview on 64-bit Arm QEMU

- مقاله How to Build and Run Android L 64-bit ARM in QEMU

به دلیل این دو موضوع تصمیم بر این شد که به جای انجام این کار، صرفا نسخه X86 اندروید ۵.۰ را بر روی QEMU اجرا نماییم. لازم به ذکر است که برای نسخه های X86 اندروید، پشتیبانی مناسبی از جانب Google و شرکت QEMU وجود دارد و مطالب زیادی در سطح اینترنت در این مورد وجود دارد.

### ۵.۰.۳ آزمایش ۴: شبیه سازی اندروید ۴.۴ نسخه پردازنده x86

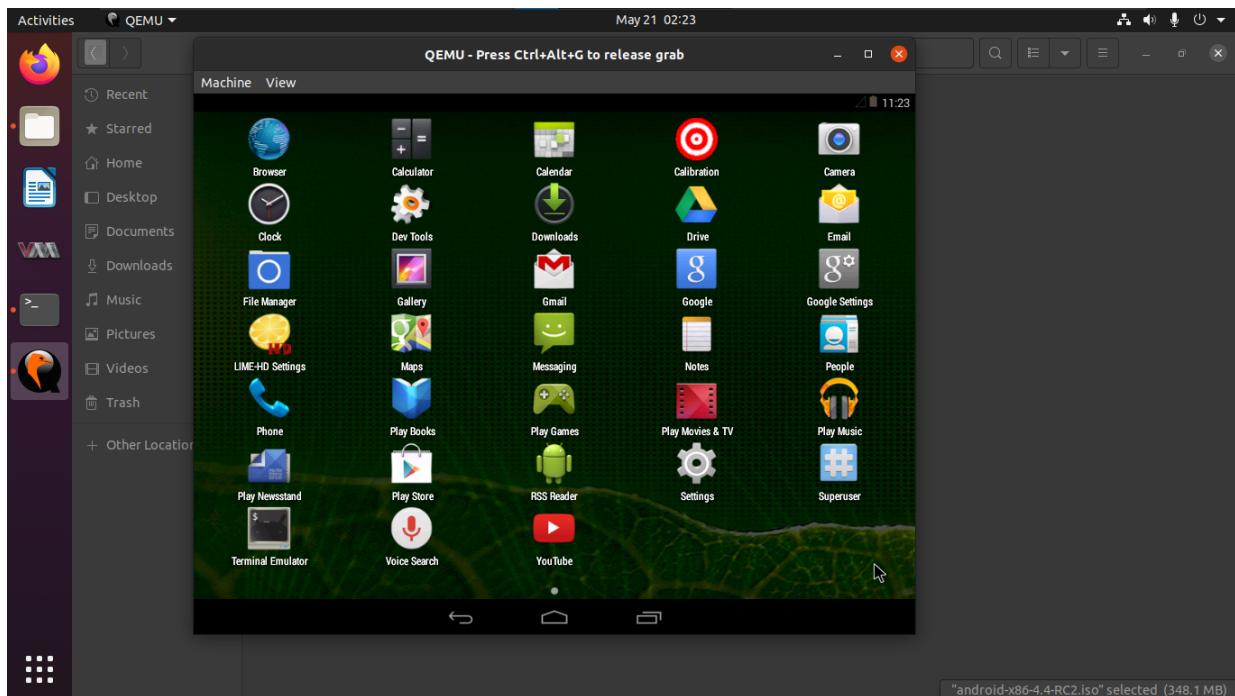
در این آزمایش قصد داریم تا نسخه ۴.۴ اندروید مخصوص پردازنده های x86 را نصب و اجرا کنیم. توجه داشته باشید که نسخه اندروید مورد استفاده در ثابت افزار کنترلر پهپاد، نسخه ۴.۴ اندروید می باشد و نصب و اجرای درست این سیستم عامل بر روی QEMU اهمیت بسیار زیادی دارد.

برای نصب، در ابتدا باید یک درایو مجازی ایجاد شود:

```
1 $ qemu-img create -f qcow2 android44_qcow2.img 8G
```

سپس لازم است که فایل اندروید از این [لینک](#) دانلود شود. پس از دانلود با اجرای این کامنده، می توان به نصب یا اجرای اندروید پرداخت:

```
1 $ qemu-system-x86_64 \
2   -vga std -m 1024 \
3   -drive file=android44_qcow2.img,cache=none \
4   -soundhw es1370 \
5   -cdrom android-x86-4.4-RC2.iso
```



شکل ۲۰: محیط اجرای اندروید ۴.۴ بر روی QEMU

## ۴ اجرا

### ۱۰.۴ دیکامپایل کردن برنامه موجود در ثابت افزار و بدست آوردن کد اندروید

در ثابت افزار ریموت کنترل و همچنین در ثابت افزار LC1860C 0901 شاهد پوشاهای به نام input.jar مشاهده می شود. در ادامه با ابزارهای مختلفی تلاش می کنیم این فایل را دیکامپایل کنیم تا به کد منبع جاوا دست پیدا کنیم. اکثر برنامه های مورد استفاده به صورت آزاد در نت در دسترس هستند و می توانید آنها را نصب و استفاده بفرمایید. راه ساده تر استفاده از ابزارهای آنلاین است که ما نیز در تلاش های یکم تا پنجم از همین ابزار استفاده کردیم که از طریق این پیوند می توانید به آنها دسترسی داشته و از آنها استفاده نمایید.



شکل ۲۱: ابزار آنلاین دیکامپایل کدهای جاوا و اندروید

#### ۱۰.۴.۱ تلاش یکم: Jadx, fast and with Android support, online

در این تلاش برنامه را به ابزار آنلاینی که jadx را بر روی ورودی اجرا می کند دادیم. این نسخه از دیکامپایلر قادر به بازیابی کامل کد اندروید نبود و کدی ابهام سازی شده را به عنوان خروجی برمی گرداند که تابع run در آن ابهام سازی شده است. در ادامه توضیحاتی در رابطه با ابهام سازی ارائه خواهد شد.

#### ۱۰.۴.۲ تلاش دوم: Procyon - fast decompiler for modern Java

به کمک ابزار آنلاین اقدام به دیکامپایل کردن کد می کنیم. متناسفانه این ابزار قادر به دیکامپایل کردن برنامه نیست و کدی به ما تحويل نمی دهد.

#### ۱۰.۴.۳ تلاش سوم: JDCore (very fast)

به کمک ابزار آنلاین اقدام به دیکامپایل کردن کد می کنیم.

متاسفانه این ابزار قادر به دیکامپایل کردن برنامه نیست و کدی به ما تحویل نمی‌دهد.

#### ۴.۱.۴ تلاش چهارم : CFR - very good and well-supported decompiler for modern Java

به کمک ابزار آنلاین اقدام به دیکامپایل کردن کد می‌کنیم.

متاسفانه این ابزار خروجی قابل توجهی به ما نمی‌دهد. فقط اعلام می‌کند که دیکامپایل انجام شده.  
تمام خروجی :

Decompiled **with** CFR 150.0

#### ۵.۱.۴ تلاش پنجم : JAD (very fast, but outdated)

به کمک ابزار آنلاین اقدام به دیکامپایل کردن کد می‌کنیم.

متاسفانه این ابزار قادر به دیکامپایل کردن برنامه نیست و کدی به ما تحویل نمی‌دهد.  
برای تلاش‌های بعدی می‌توان به سراغ دیکامپایلر karakatua رفت.

#### ۶.۱.۴ تلاش ششم (موفق) : Jadx, fast and with Android support on local machine

برخلاف تلاش یکم، این بار ابزار jadx را نصب و اجرا می‌کنیم و کد منبع با موفقیت بدست می‌آید.

برای بارگیری و نصب به مخزن گیت‌هاب **jadx** مراجعه کنید.

با گسترده‌سازی فایل m2rc-dump.zip وارد پوشه system/framework می‌شویم. فایلی به نام input.jar را به کمک ابزار jadx با دستور

```
1 ./. jadx address-to-project/system/framework
```

دیکامپایل می‌کنیم و کد منبع به زبان جاوا/اندروید به دست می‌آید.

مشاهده اولیه ما این است که کنترلر به این صورت کار می‌کند که انواع عملیات‌هایی که می‌شود با دسته انجام داد و دکمه‌هایی که می‌شود کلیک کرد را به کمک کتابخانه‌های اندروید inject .می‌کند (ارسال به کواد کوپیتر)

در بخش ۲.۴ به توضیح بخش‌های مختلف کد بدست آمده خواهیم پرداخت. فایل‌هایی که به آن‌ها اشاره شده است در مخزن گیت‌هاب پروژه به همراه سورس کد موجود هستند.  
خلاصه‌ی خروجی بدست آمده:

```
1 * summery *
```

```
2
```

```
3 Input:
```

```
4
```

```
5 Files:
```

```

⁹
⁺ + rc_dump/system/framework/input.jar
⁻
⁹ Code sources:
¹⁰
¹¹ + classes.dex
¹²
¹³ Counts:
¹⁴
¹⁵ + Classes: 2
¹⁶ + Methods: 15
¹⁷ + Fields: 3
¹⁸ + Instructions: 1024 (units)
¹⁹
²⁰ Decompilation:
²¹
²² + Top level classes: 1
²³ + At process stage: 0 (00.0%)
²⁴ + Code generated: 0 (00.0%)
²⁵
²⁶ Issues:
²⁷
²⁸ + Errors: 0
²⁹ + Warnings: 0
³⁰ + Nodes with errors: 0
³¹ + Nodes with warnings: 0
³² + Total nodes with issues: 0
³³ + Methods with issues: 0
³⁴ + Methods success rate: 00.100%

```

## ۲.۴ تحلیل و تشریح کد اندروید بدست آمده از ثابت افزار

در این بخش به بررسی خروجی ابزار Jadx می‌پردازیم.

### ۱۰.۴ ابهام‌سازی یا Obfuscation

به این بخش از کامنت آورده شده در خروجی تلاش یکم توجه کنید:

```

¹ JADX WARNING: Code restructure failed: missing block: B:40:0x00e4, code lost:
² sendSwipe(r1,

```

```

۱     java.lang.Float.parseFloat(r15[r9 + 1,])
۲     java.lang.Float.parseFloat(r15[r9 + 2,])
۳     java.lang.Float.parseFloat(r15[r9 + 3,])
۴     java.lang.Float.parseFloat(r15[r9 + 4,])
۵     r6);
۶ JADX WARNING: Code restructure failed: missing block: B:59,:? code lost:
۷ return;
۸ Code decompiled incorrectly, please refer to instructions dump.

```

با وجود اینکه بخش هایی از کد جاوا دیکامپایل شده و قابل مشاهده است، اما با این وجود ابزار Jadx موفق به دیکامپایل کامل و بی نقص برنامه نشده است.

مشخصاً این برنامه بهنگام تبدیل به فایل jar تحت نوعی از ابهام سازی قرار گرفته تا امکان دیکامپایل کامل آن فراهم نباشد.

ابهام سازی تغییراتی است که در کد برنامه ایجاد می شود به صورتی که کارکرد و منطق کد دست نخورده باقی می ماند اما صورت و نمایش آن به گونه ای تغییر می کند که خوانش کد برای انسان بسیار دشوار و تقریباً غیرممکن شود. همچنین با این اقدام می توان دیکامپایلرهایی را گمراه کرد تا نتوانند بازیابی کاملی از کد اولیه داشته باشند. برای مثال کاری که انجام می شود این است که نام تمامی متغیرها، توابع و مدل ها به اسمی بی معنی تغییر داده می شود یا یک حلقه به چند حلقه شکسته می شود یا اقداماتی نظری این ها.

## ۲۰۰۴ توابع دیکامپایل شده در تلاش ششم

- public class Input •

کلاسی از اشیا که می توانند دارای مقادیر مختلف ورودی های قابل ارسال توسط کنترل از راه دور باشند.

- private void run(String[] args) •

این تابع در واقع تابع اصلی اجرای برنامه است. در این تابع با توجه به ورودی عمل انجام شده روی کنترلر را تشخیص می دهیم و با توجه به آن از تابع داخلی متناسب استفاده می کنیم تا اطلاعات مورد نیاز برای کواد کوپتر ارسال شوند.

این توابع ثانویه هر کدام در خود با یک دستور inject کلید یا حرکت دریافت شده را به کواد کوپتر ارسال می نمایند.

- private void sendMove(), sendSwipe(), sendTap(), sendText(), sendKeyEvent() •

یکی از توابع injectMotionEvent و injectKeyEvent را با مقادیر مناسب صدا می کنند.

- private void injectMotionEvent() •

یک MotionEvent را توسط InputManager که از کتابخانه های آماده اندروید است inject می کند. همچنین در LOG انجام این مورد را ثبت می کند.

private void injectKeyEvent() •

همانند تابع قبلی فقط بجای KeyEvent یک MotionEvent را inject می کند. همچنین در LOG این مورد را ثبت می کند.

private void showUsage() •

انواع و اقسام دستوراتی که این کلاس از آنها پشتیبانی می کند را چاپ و معرفی می کند.

## ۳.۴ تحلیل firmware با ابزار firmwalk

تجزیه و تحلیل دستی ثابت افزار نیازمند دققت و زمان زیادی است و به دلیل محدودیت های زمانی اغلب نمی توان آنالیز دستی انجام داد. بنابراین، تجزیه و تحلیل خودکار ثابت افزار بسیار کمک کننده خواهد بود. ما برای این کار از ابزار firmwalker استفاده کردیم که ابزاری برای تحلیل firmware است و از این [لينك](#) قابل دسترسی می باشد. firmwalker در واقع یک bash script است که موارد زیر را شناسایی می کند.

- ۱ etc/shadow **and** etc/passwd
- ۲ **list** out the etc/ssl directory
- ۳ search **for** SSL related files such **as** .pem, .crt, etc.
- ۴ search **for** configuration files
- ۵ look **for** script files
- ۶ search **for** other .bin files
- ۷ look **for** keywords such **as** admin, password, remote, etc.
- ۸ search **for** common web servers used on IoT devices
- ۹ search **for** common binaries such **as** ssh, tftp, dropbear, etc.
- ۱۰ search **for** URLs, email addresses, **and** IP addresses
- ۱۱ Experimental support **for** making calls to the Shodan API using the Shodan CLI

برای استفاده کافی است ثابت افزار را از حالت فشرده خارج کنیم (برای نتیجه مطمئن تر می توان از binwalk برای گستردگی سازی استفاده کرد) و آن را در مسیر firmwalker/firmwalker-master قرار دهیم و همان جا دستور زیر را اجرا کنیم. در نتیجه تحلیل یک فایل متند به فرمت txt. در همان آدرس نوشته خواهد شد. که شامل اطلاعات مذکور است.

۱ \$ ./firmwalker.sh firmwareName.extracted

```
***Search for shell scripts***  

##### shell scripts  

p/system/bin/adb_en.sh  

p/system/bin/aging_test.sh  

p/system/bin/check_1860_state.sh  

p/system/bin/check_cp_status.sh  

p/system/bin/check_lmi2_state.sh  

p/system/bin/check_scsi.sh  

p/system/bin/config_soc_pwm0_for_fan.sh  

p/system/bin/dji_log_all.sh  

p/system/bin/dji_log_control.sh  

p/system/bin/dji_net.sh  

p/system/bin/efuse_dump_check.sh  

p/system/bin/efuse_enck.sh  

p/system/bin/efuse_step1.sh  

p/system/bin/efuse_step2.sh  

p/system/bin/factory_out_test.sh  

p/system/bin/get_test_result.sh  

p/system/bin/in_whitelist.sh  

p/system/bin/kernel_profiled_debug.sh  

p/system/bin/lib_env.sh  

p/system/bin/lib_test.sh  

p/system/bin/lib_test_220rc.sh  

p/system/bin/lib_test_cases.sh  

p/system/bin/lib_test_stress.sh  

p/system/bin/lib_test_utils.sh  

p/system/bin/modem_info.sh  

p/system/bin/otp_flag.sh  

p/system/bin/panic_tombstone_check.sh  

p/system/bin/parse_logcat.sh  

p/system/bin/part_check.sh  

p/system/bin/recovery_update.sh  

p/system/bin/reset_rc_user_config.sh  

p/system/bin/secure_debug.sh  

p/system/bin/setup_usb_serial.sh  

p/system/bin/set_test_result.sh  

p/system/bin/set_time.sh  

p/system/bin/speaker_cali.sh  

p/system/bin/speaker_leaky.sh  

p/system/bin/start_dji_system.sh  

p/system/bin/stress_test.sh  

p/system/bin/test_boardsn.sh  

p/system/bin/test_buzzer.sh  

p/system/bin/test_cp_gnd.sh  

p/system/bin/test_cp_reset.sh  

p/system/bin/test_enck.sh  

p/system/bin/test_fan.sh  

p/system/bin/test_flash.sh  

p/system/bin/test_mcu_link.sh  

p/system/bin/test_mcu_uart.sh  

p/system/bin/test_mem.sh  

p/system/bin/test_mp_stage.sh  

p/system/bin/test_multi_enc.sh  

p/system/bin/test_ota.sh  

p/system/bin/test_rc_key.sh  

p/system/bin/test_rc_lcd.sh  

p/system/bin/test_rc_link.sh  

p/system/bin/test_rc_usb.sh  

p/system/bin/test_sdr.sh  

p/system/bin/test_speaker.sh  

p/system/bin/test_thermal.sh  

p/system/bin/test_vibrator.sh  

p/system/bin/test_video_dump.sh  

p/system/etc/install-recovery.sh
```

شکل ۲۲: بخشی از خروجی ابزار تحلیل ثابت افزار برای تحلیل کنترلر

موارد یافت شده در خروجی برنامه برای ثابت افزار کنترلر:

```

1 *.conf
2 *.cfg
3 shell scripts
4 patterns::token, gpg, pgp, secret, telnet, private key, ssl,\\
5 pwd, passwd, password, root, admin, upgrade
6 busybox
7 ip addresses
8 urls
9 emails

```

## ۴.۴ اجرای ثابت افزار مربوط به کنترلر بر روی QEMU

در این بخش، برای تحلیل بهتر و مناسبتر ثابت افزار کنترلر، قصد داریم تا با آزمایش های مختلف، به اجرای این ثابت افزار بر روی شبیه ساز بپردازیم. در ابتدا با ساختار کلی فایل نصی اندروید آشنا می شویم. سپس فایل system.sfs که یکی از مهم ترین فایل های موجود در iso اندروید می باشد را بررسی می کنیم. و درنهایت سه آزمایش مختلف جهت اجرای ثابت افزار انجام می دهیم.

### ۱۰.۴.۴ ساختار کلی فایل نصی اندروید

ساختار فایل نصی iso اندروید به شکل زیر می باشد:

```

1 /.disk
2 /boot
3 /efi
4 /isolinux
5 /[BOOT]
6 initrd.img
7 install.img
8 kernel
9 ramdisk.img
10 system.sfs
11 TRANS.TBL

```

توجه داشته باشید که ممکن است فایل ها و یا دایرکتوری های دیگری با توجه به نسخه اندروید در فایل iso موجود باشد و یا برخی از فایل ها و دایرکتوری ها در iso قرار نگرفته باشند.  
از بین دایرکتوری ها و فایل های موجود، دایرکتوری boot و فایل system.sfs و همچنین kernel اهمیت بیشتری نسبت به بقیه دارند. در دایرکتوری boot کانفیگ های مربوط به بوت لینوکس قرار دارد و عملاً بدون این دایرکتوری، سیستم عامل اجرا نمی شود.  
kernel نیز هسته و اصلی ترین بخش سیستم عامل است که پایه ای ترین سرویس ها برای سایر بخش های سیستم عامل را مهیا می کند.

فایل system.sfs نیز که یک فایل فشرده می باشد، شامل فایل ها و دایرکتوری های اصلی سیستم عامل است. برای اجرای ثابت افزار، لازم است که به گونه ای فایل ها و دایرکتوری های آن را به صورت system.sfs در بیاوریم و آن را کنار سایر فایل های مورد نیاز در iso قرار دهیم و فایل نصی جدید را ایجاد کنیم. در ادامه چگونگی انجام این مراحل را بررسی می کنیم.

#### ۲۰۴.۴ نحوه ایجاد فایل system.sfs

ساختار فایل system.sfs به اینگونه است که در داخل آن یک فایل فشرده با نام system.img وجود دارد که داخل این فایل، فایل سیستم اصلی سیستم عامل قرار دارد. نکته قابل توجه این است که در فایل دامپی که ما برای ثابت افزار در اختیار داریم، شامل یک دایرکتوری با نام system است که فایل سیستم اصلی ثابت افزار را دارا می باشد و با تبدیل این دایرکتوری به system.img و سپس system.sfs، این دایرکتوری را آماده اجرا خواهیم کرد.

Name	Size	Packed Size	Modified	Accessed	Folders	Files
app	134 004 610	134 098 944	2014-05-20 16:45	2022-05-26 00:44	0	89
bin	8 871 357	8 862 048	2014-05-20 16:39	2022-05-26 00:44	0	100
etc	1 111 594	1 359 872	2014-05-20 16:45	2022-05-26 00:44	10	205
fonts	18 583 848	18 659 328	2014-05-20 15:56	2022-05-26 00:44	0	72
framework	55 651 531	55 795 712	2014-05-20 16:44	2022-05-26 00:44	2	127
lib	284 288 391	286 896 128	2014-05-20 16:44	2022-05-26 00:44	532	2 611
lost+found	0	0	2014-05-20 16:45	2022-05-26 00:43	0	0
media	1 015 393	1 056 768	2014-05-20 15:09	2022-05-26 00:44	5	36
priv-app	95 698 767	95 762 432	2014-05-20 16:44	2022-05-26 00:44	0	64
usr	42 717 684	42 817 536	2014-05-20 15:35	2022-05-26 00:44	13	91
vendor	15 001 296	15 013 888	2014-05-20 16:36	2022-05-26 00:44	8	10
xbin	3 451 808	3 471 360	2014-05-20 16:38	2022-05-26 00:44	0	24
[SYS]	11 534 336	11 534 336	2014-05-20 16:45	2022-05-26 00:43	0	1
build.prop	2 325	4 096	2014-05-20 14:29	2022-05-26 00:43		

شکل ۲۳ : ساختار داخلی system.sfs

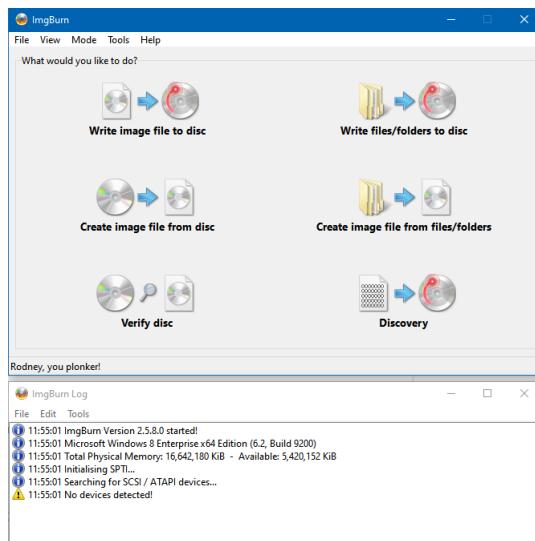
برای ایجاد فایل img، از نرم افزار [ImgBurn](#) استفاده می کنیم و برای ایجاد فایل فشرده sfs از دو روش می توان استفاده کرد:

- استفاده از ابزار RMXTools v1.5 و بر اساس راهنمایی این [ویدیو](#)

- با استفاده از ابزار SquashFS اویونتو و کامند mksquashfs

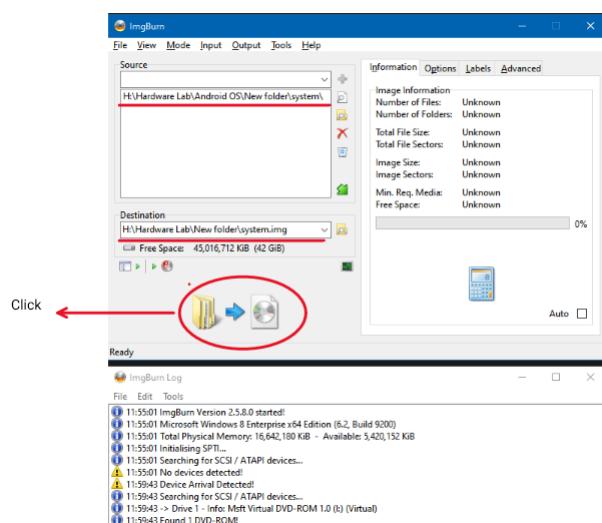
در ادامه قصد داریم نحوه ایجاد فایل iso و img با استفاده از نرم افزار ImgBurn را بررسی می کنیم و سپس نحوه ایجاد system.sfs را با هر دو روش انجام می دهیم.

۱۰۲۴۴ استفاده از ابزار **IMGBurn** برای ایجاد فایل **img** و **iso** نرم افزار **IMGBurn** یکی از نرم افزارهای معروف در زمینه ایجاد فایل های فشرده مناسب برای انواع دیسک های مختلف می باشد. از این نرم افزار برای ایجاد فایل فشرده به فرمت **img** استفاده خواهیم کرد.



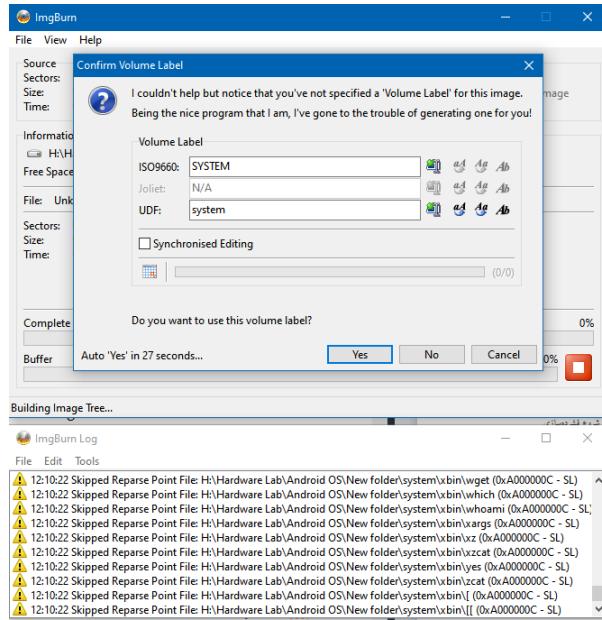
شکل ۲۴: تصویری از محیط IMGBurn

دایرکتوری system را قصد داریم تا با استفاده از نرم افزار IMGBurn به فایل فشرده **img** تبدیل می کنیم.

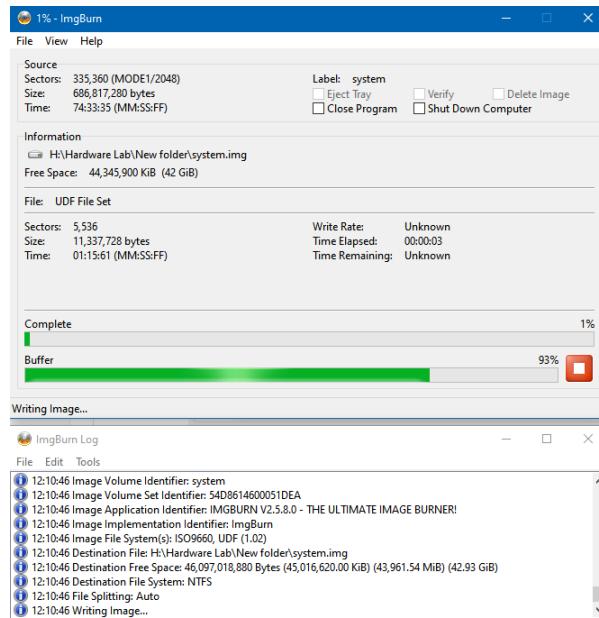


شکل ۲۵: انتخاب آدرس src و شروع فشرده سازی destination

پس از انتخاب دایرکتوری مبدا و فایل مقصد، فرآیند فشرده سازی را اجرا می کنیم. چندین پیام مختلف در نرم افزار ظاهر می شود و کافی است در تمامی آنها Yes زده شود. پس از این پیام ها، فرآیند فشرده سازی اجرا می گردد.

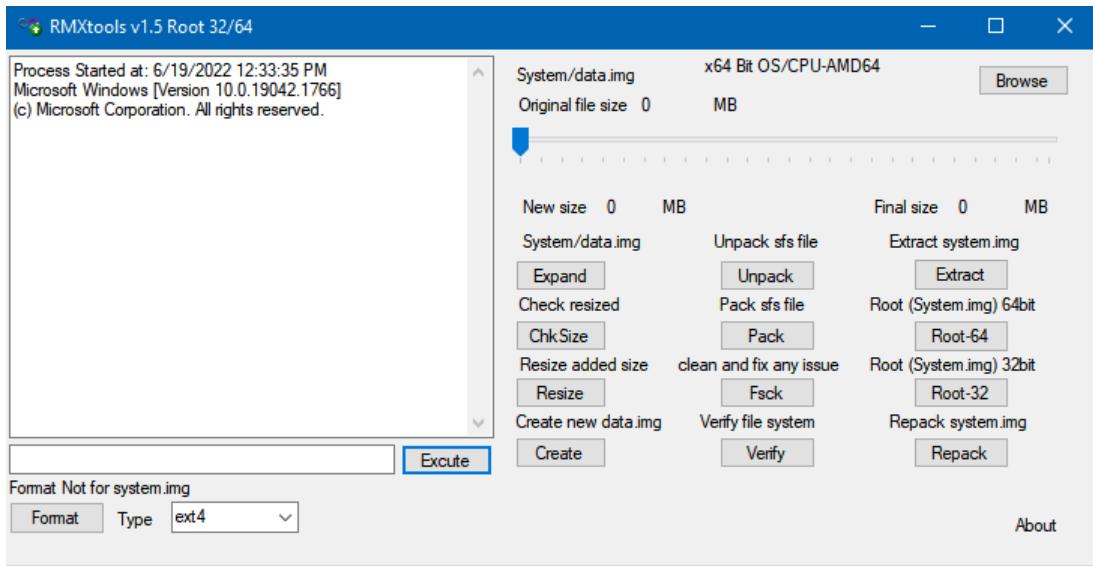


شکل ۲۶: مشخصات فایل فشرده



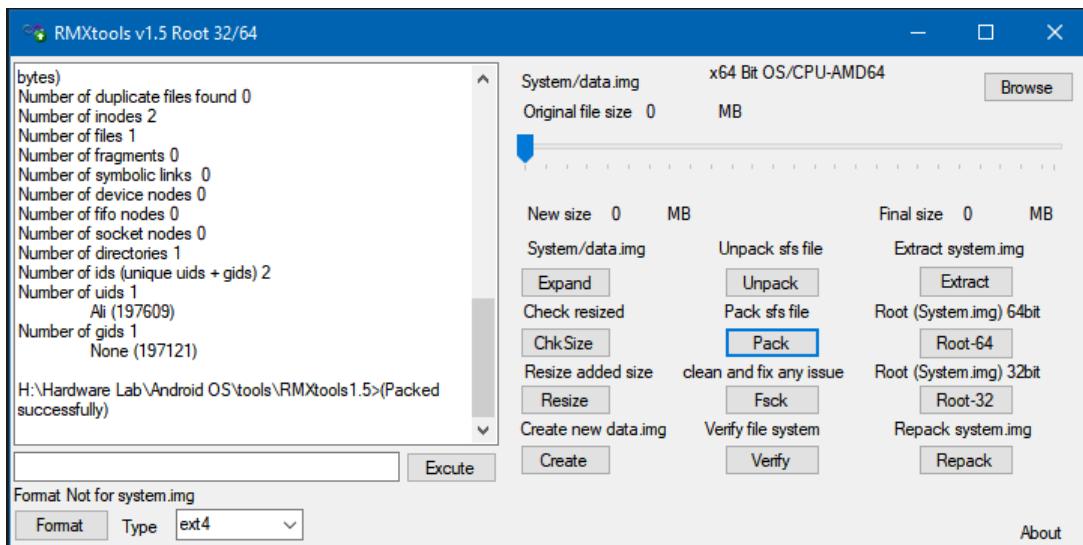
شکل ۲۷: اجرای فشرده سازی

**۲۰۲۰۴۰۴ استفاده از ابزار RMXTools** از نرم افزار RMXTools برای روت کردن سیستم عامل Remix OS (که نوعی اندروید قابل اجرا بر روی معماری X86 می باشد) استفاده می گردد. از امکاناتی که این نرم افزار دارد، ایجاد فایل فشرده system.sfs برای فایل iso اندروید می باشد. در ابتدا باید نسخه ۱.۵ این نرم افزار را نصب کرد. برای این کار باید فایل نرم افزار را از طریق این [لینک](#) دانلود کرد و سپس به نصب آن پرداخت.



شکل ۲۸: تصویری از محیط RMXTool

برای ایجاد فایل system.sfs باید در ابتدا فایل فشرده system.img را ایجاد کرده و سپس با استفاده از گزینه file Pack sfs به ایجاد فایل system.sfs بپردازیم.



شکل ۲۹: تصویری از محیط RMXTool بعد از ایجاد system.sfs

۳.۰.۴.۴ استفاده از ابزار SquashFS **SquashFS** یک نوع فایل سیستم در لینوکس می‌باشد که حالت دارد و فایل با تایپ sfs به نوعی از این نوع فایل سیستم می‌باشد. برای ایجاد

با استفاده از این فایل سیستم از کامند system.sfs

```
$ mksquashfs /src_dir /target_file.sqsh
```

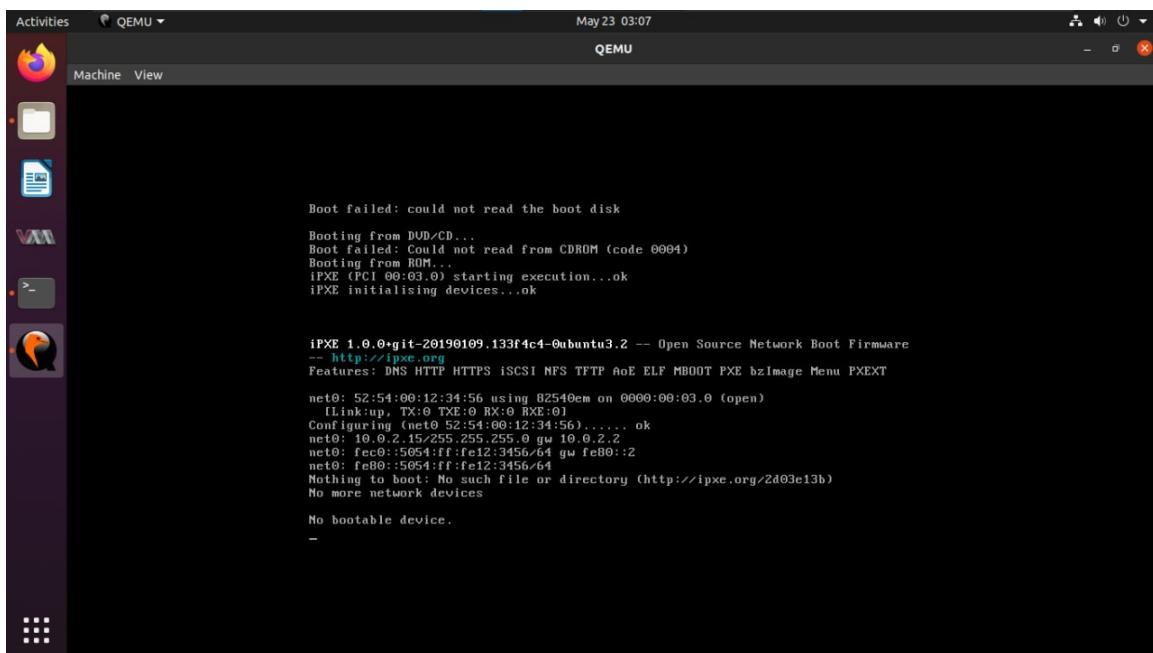
در ادامه کار، با انجام چند آزمایش قصد داریم تا اجرای ثابت افزار را شبیه سازی کنیم.

#### ۳۰.۴.۴ آزمایش ۱: شبیه سازی اندروید ۴.۴ نسخه پردازنده x86

اجرای این نسخه اندروید شبیه اجرای اندروید ۸.۱ است. صرفا لازم است که با استفاده از این [لينک](#) به دانلود فایل اندروید می پردازیم، سپس با استفاده از کامند زیر، به اجرای سیستم عامل مانند بخش ۵.۲.۳ می پردازیم.

#### ۴۰.۴.۴ آزمایش ۲: شبیه سازی نسخه دستکاری شده اندروید ۴.۴ با استفاده از فایل های دامپ

در ابتدا فایل ها و دایرکتوری های موجود در دامپ ثابت افزار کنترلر را به system.sfs تبدیل می کنیم و فایل sfs جدید را جایگزین system.sfs قبل در فایل نصبی اندروید قرار می دهیم. سپس به اجرای اندروید تغییر داده شده می پردازیم.

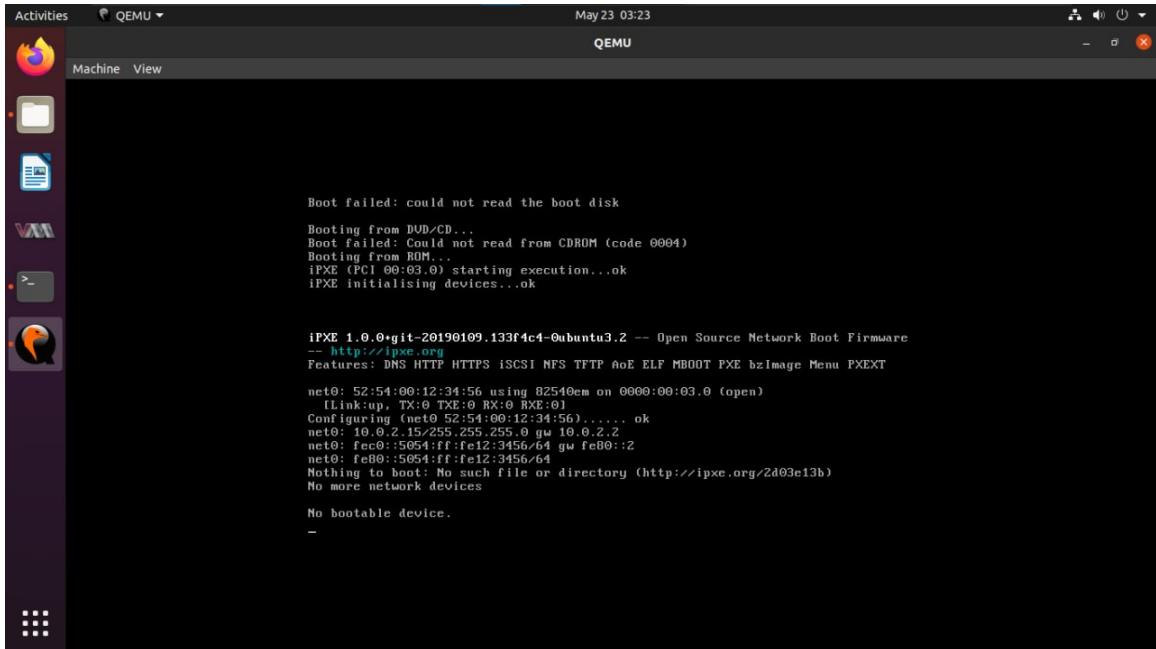


شکل ۳۰: نتیجه اجرای آزمایش ۲

همانگونه که در تصویر ۳۰ مشاهده می کنید، اجرای این آزمایش به خطا می خورد. علت خطا نیز این است که سیستم با استفاده از CD/DVD بوت نمی شود و سیستم نمی تواند اجرا شود. توجه داشته باشید که با هر دو روش ارائه شده در بخش قبل به با هردو روش ذکر شده، فایل sfs ایجاد می گردد، اما در هر دو روش، به مشکل یکسان برخورد می کنیم.

### ۵.۰.۴ آزمایش ۳: شبیه سازی نسخه دستکاری شده اندروید ۴.۴ با استفاده از ترکیب فایل های دامپ و اندروید

برای این آزمایش، فایل ها و دایرکتوری های موجود در دامپ به صورت replace در دایرکتوری system در QEMU قرار گیرد. کپی می کنیم و سپس فایل system.sfs را در محتويات فایل iso ایجاد می کنیم و سپس فایل نصبی جدید را ایجاد و اجرا می نماییم.



```

Activities QEMU May 23 03:23
Machine View QEMU
Boot failed: could not read the boot disk
Booting from DVD/CD...
Boot failed: Could not read from CDROM (code 0004)
Booting from ROM...
iPXE (PCI 00:03.0) starting execution...ok
iPXE initialising devices...ok

iPXE 1.0.0+git-20190109.133f4c4-0ubuntu3.2 -- Open Source Network Boot Firmware
-- http://ipxe.org
Features: DNS HTTP HTTPS iSCSI NFS TFTP AoE ELF MBOOT PXE bzImage Menu PXEXT

net0: 52:54:00:12:34:56 using 82540em on 0000:00:03.0 (open)
[Link up, TX:0 RX:0 BXE:0]
Configuring (net0 52:54:00:12:34:56)... ok
net0: 10.0.2.15/255.255.255.0 gw 19.0.2.2
net0: fec0::5654:ff:fe12:3456/64 gw fe80::2
net0: fe80::5654:ff:fe12:3456/64
Nothing to boot: No such file or directory (http://ipxe.org/2d03e13b)
No more network devices

No bootable device.
-
```

شکل ۳۱: نتیجه اجرای آزمایش ۳

در اجرای این قسمت نیز با مشکل بخش قبل مواجه می شویم.

## ۵ نتیجه‌گیری

در این پروژه تلاش شد تا با انجام تحقیقات درباره اجزای پهپاد و ساختار فایل ثابت افزار و همچنین تحلیل و بررسی بخش‌های مختلف ثابت افزار به همراه اجرای آن بر روی شبیه‌ساز، عملکرد ثابت افزار کنترلر و کواد کوپتر پهپاد تجاری 2 DJI Mavic Pro بررسی شود.

در تحقیقات انجام شده، سخت افزار پهپاد و کنترلر به طور کامل بررسی شد و همچنین فایل سیستم اندروید بررسی گردید. سپس لینک‌ها، ریپازیتوری‌ها و منابع مفید معرفی گردید. در بخش شبیه‌ساز نیز، در مورد شبیه‌سازهای مختلف پردازنده ARM بررسی و تحقیق انجام گردید و آزمایش‌های مختلفی برای آشنایی بیشتر با شبیه‌ساز QEMU ترتیب داده شد.

در بخش اجرا نیز سعی شد تا با استفاده از اطلاعات و ابزارهایی که در اختیار است، به اجرا و تحلیل بیشتر ثابت افزار پردازیم. برای راحتی کار نیز، تنها ثابت افزار مربوط به کنترلر پهپاد بررسی گردید. در نهایت نیز با توجه به برخی از مشکلات و چالش‌ها، اجرای ثابت افزار بر روی شبیه‌ساز با مشکل مواجه شد و نتوانستیم شبیه‌ساز را به طور کامل بر روی شبیه‌ساز اجرا کنیم. البته نکته قابل توجه این است که با سرچ و پرس‌جو در سطح اینترنت و فروهمان و کامیونیتی‌های مربوطه، کسی تاکنون اقدام به اجرای ثابت افزار بر روی شبیه‌ساز نکرده بود و راهنمای جامع و قابل اعتمادی برای انجام این کار وجود نداشت.

علاوه بر تلاش برای اجرای ثابت افزار بر روی شبیه‌ساز، بخش‌های مختلف دامپ ثابت افزار بررسی گردید. یکی از این موارد، بررسی فایل jar موجود در یکی از دایرکتوری‌های دامپ بررسی گردید و با استفاده از ابزار jadx این فایل دیکامپایل گردید و کد جاوای مرتبط با عملیات‌های قابل انجام با دکمه‌های کنترلر استخراج گردید. همچنین با استفاده از ابزار firmwalk بخش‌های مرتبط دامپ بررسی و تحلیل گردید.

گام‌های بعدی برای بررسی و تحلیل ثابت افزار، استفاده از روش‌های دیگر و جهت‌های دیگر برای بررسی فایل‌های ثابت افزار است. برای مثال می‌توان ثابت افزار را بر روی چیپست مناسب لود کرد و با استفاده از روش‌های مبتنی بر بررسی و تحلیل سخت افزار، به تحلیل و بررسی عملکرد ثابت افزار پرداخت.